Learning Radial Basis Function Based Soccer Strategies using Ideal opponent Model

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by
Sreangsu Acharyya

under the guidance of
Dr Amitabha Mukerjee.

to the
Department of Mechanical Engineering
Indian Institute of Technology, Kanpur **MARCH 2000**

Certificate

This is to certify that the work contained in the thesis entitled **Learning Radial Basis Function Based Soccer Strategies using Ideal opponent Model** has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

[illegible]

Dr. Amitabha Mukerjee

Associate Professor,
Department of Computer Science,
Indian Institute of Technology,
Kanpur.

Contents

List of Figures

List of Tables

List of Notations

$\mathcal{S}_t$	State at time t.
$\mathcal{S}'$	Any valid state.
$\mathcal{A}_i$	Action i.
$\mathcal{T}$	Transition Probability.
$\pi(\mathcal{S})$	Action selection policy.
γ	Dicount rate.
r_t	Reward obtained at time t.
α	Learning rate.
$\mathcal{Q}$	Action value function.
$\mathcal{Q}^*$	Optimal action value function.
$\hat{\mathcal{Q}}$	Predicted action value function.
$\mathcal{V}$	State value function.
$\mathcal{V}^*$	Optimal state value function.
$\overline{x}$	State vector input of RBF.
$\overline{c}$	Centre vector of RBF.
σ	Width of RBF unit.
Σ	Width matrix (covariance matrix).

f_i	RBF function.
F	Matrix of RBF outputs.
w	Weights.
Φ	New data vector.
E, e	Error.
σ^2_{MSE}	Mean square error.
σ^2_{LOO}	Leave one out Cross-validation.
σ^2_{GCV}	Generalised Cross-validation.
λ	Lagrange multipliers.
μ	Membership value.
$\mathcal{N}$	Uniformly distributed noise.
$K()$	Kernel function.
ϵ	Threshold value.
n_{eff}	Effective number of RBF units.
$d_s, d_g, d_{d1}, d_{d2}, d_b, d_o$	Reciprocal of the distance of various players on the field.
$\theta_s, \theta_g, \theta_{d1}, \theta_{d2}, \theta_b, \theta_o$	Angle coordinates of various players on the field.
β	Shooting angle parameter.
P, p	Probability.

Acknowledgement

The objective style of of academic writing hardly allows one to be personal in his writings, except in this section. So here I take the this liberty to thank everyone without whom the work would not have been as enjoyable.

The first place in the acknowledgement section, is always reserved for the student's advisor for giving a helping hand and a guiding direction. I would like to thank Dr Amitabha Mukerjee for his constant stream of "positive reinforcements" and advice through out the course of this work. However what was even more important to me, was that it was fun. Sir thanks for that.

Thanks goes to Dr. Bhaskar Dasgupta for his stimulating and enjoyable introduction into the theories of robotics and may I add: letting me off lightly in terms of TA duties.

Thanks to Swaroop and all my class-mates for tolerating my incessant questions and requests. The Centre of Robotics Laboratory would not have been same without Nishant's disk jokeying and Tamhant supply of chocolates.

Special thanks to Prof B. C. Raymahashay and Prof N. N. Kishore, without whose help my application would never have reached IIT kanpur in time.

My stay here would not have been as memorable without the illustrious tikka, biriayni, kaleji, badnam and lassi fellows of *ie*^{3} *iitk* and their activities. Not to mention the late night music sessions in our hall which must have denied many hall mates of their rare and precious hours of sleep. Finally thanks to all those obvious people, whom i find embarassing to mention. Thanks for keeping my spirirt up through mails and phone calls.

Abstract

Reinforcement Learning methods having the ability to learn from "evaluative" feedback forms a viable alternative to Supervised Learning. Especially so, when "state-optimal action pairs" are costly or impossible to generate (because of lack of domain knowledge or the requirement of online performance). Algorithms with proved convergence to optimality exists for discrete state space models i.e. with table look up architectures. Application of Reinforcement Learning to large or continuous valued state spaces, however, necessitates the use of function approximators like Radial Basis Networks and Neural Networks, albeit at the cost of loss of proof of optimality.

This study investigates the applicability of radial basis function based reinforcement learning methods in a dynamic multiagent scenario of robot soccer. Radial Basis Function Networks poses as an attractive method of function approximation for these tasks. The advantages over more popular feed forward sigmoidal Neural Networks being that they can be interpreted as fuzzy rule based systems thereby offering comprehensible as opposed to black-box ANN models, they are much faster to train, are analytically incremental and being local in their response are more suited for reinforcement learning tasks than ANN. Opponent modelling, a factor so far neglected in this particular domain has been incorporated through the presence of opponents assumed to act optimally. The application of RBF has been facilitated by expediting the recall time through the development of a new tree structured implementation named RBF.

Chapter 1

Introduction

This work considers a multi-agent, dynamic game scenario, where autonomous agents have to act in real time, in a fast changing, noisy environment. Moreover in domains like these e.g. robot soccer, the result of an agent's action may be explicit only at a future time. Learning algorithms, have enabled researchers to handle these complex domains, without the need for modelling the optimal policy explicitly.

Learning algorithms can be divided into two broad categories supervised learning and reinforcement learning. Supervised learning is a methodology by which an agent can learn from "*examples*", as opposed to performing on the basis of pre-formulated explicit policies. Supervised Learning requires an expert to generate a training set in the form of "(state, action)" pairs, that are optimal. Thus supervised learning cannot be used to learn directly from the environment.

An alternative would be to use the feedback from the environment for the learning task. Environments, however seldom provide "instructions" of the optimal actions. A more realistic scenario would be were the environment provides an evaluation of the applied action in the form of rewards and penalties. Such learning tasks can be addressed by the method of Reinforcement learning, which can learn from "*experience*".

Many of the Reinforcement Learning algorithms are based on look up table representation of the state. Real life problems often involve huge state spaces, which makes table based state representation impossible. Some compaction might be obtained by replacing tabular lookup by function approximators, but even then computational requirements far exceed available resources. A possible work around as introduced by Brooks [1], is to consider

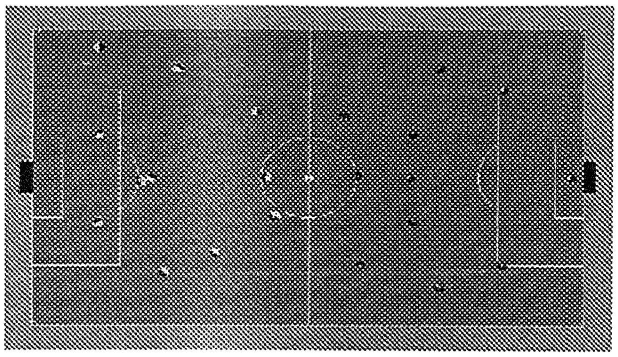


Figure 1.1: soccer server screen

the task to be composed of a hierarchy of simpler behaviours. Such artificial, hierarchical segmentation is expected to introduce sub-optimality, but its advantage lies in making impossible problems tractable.

1.1 Learning in soccer domain:

Robot-Soccer has emerged as an interesting and highly active domain for several artificial intelligence paradigms especially learning multi-agent behaviours [2]. The current work deals with the simulation league, where a standardised software - "soccer-server" simulates the physics of the game by specified stochastic state update rule. The participants are required to design the behaviours of individual agents which have to base their action upon the state as obtained from the soccer-server. The size of the state space at a resolution of 1m and 1 degree is of the order of 10^{143}. [1]

Learning task in this domain has mainly utilized feed-forward neural networks in the role of supervised learning. This leads to methods which are off-line, thus unable to adapt to opponents. Being black-box models they are difficult for a human to understand. The requirements for a supervised learning algorithm are also much higher because of its need of optimal state -action pairs.

Among the different tasks learnt in this domain include simple kicking to empty goal [3] using feed-forward neural networks. The network was used to learn when to start accelerating to hit an approaching ball into the goal. Though the paper touches on the need for adversarial learning, where the goal is defended by a keeper, the authors did not train a neural network

[1]The size of the field is $100m \times 65m$ and the total number players is 22.

non-reactive defender circling in front of the goal in a fixed trajectory, by using memory based learning techniques. The learning consisted of choosing between two options shooting directly to the centre of the goal or passing to a team mate at a fixed position who took the goal shot.

Decision tree induction (C4.5) was used by the same authors [5] to predict whether a team-mate could receive a pass from a fixed location. The task of unchallenged interception was learnt by the help of feed-forward neural networks in [6]. The same decision tree inducing software C4.5 was used by Tambe and his co researchers [7] to learn the optimum shooting angle from data generated by human experts. Up to 3 kicking solutions were considered namely at the sides and the centres.

It is only very recently that online training has been executed in practice [7]. A mixed offline-online strategy was used to train a interception routine. The opponents were not considered explicitly. However, because the training was online, the individual agents could implicitly adapt to the behaviours of nearby opponents.

1.2 Approach

In this work Radial Basis Function Networks were used as function approximators for Learning of two low level skills, developed on the assumption of optimally acting opponents:

1. adversarial shooting to goal.

2. adversarial interception.

The advantages of RBF networks include that they can be interpreted as fuzzy rule based system, having closed form expression for the learnt parameters they are much more faster to train than feedforward sigmoidal neural networks, more importantly they can be updated online with guaranteed preservation of past training cases. RBF methodology gives a strong linear algebraic foundation to the field of fuzzy inferencing, thereby introducing a number of tools for tuning its parameters. The validity of convergence of most Reinforcement Learning algorithms depend on tabular state representation and fail to remain true when function approximators are used. This is primarily because of over generalisation over the input space, to which ANN

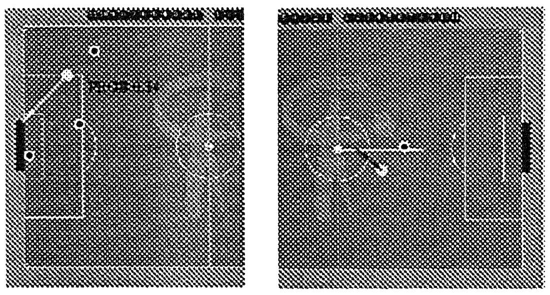


Figure 1.2: The agent in light coloured circle tries to shoot to goal in presence of opponents shown with a light ring around a dark interior. The Light arrow in the left shows the prescribed direction of shooting (left).
On the right the agent tries to intercept a moving ball signified by the light arrow, in presence of opponents. The end of the arrow shows the point the ball can reach in 10 time steps. The prescribed trajectory is shown by the dark arrow.

are typically prone. RBF being local models is largely free of such limitations and are more suitable for Reinforcement Learning tasks.

The use of Radial basis function networks which incorporates the comprehensibility of a fuzzy rule as well as a probabilistic interpretation, has not been seen to be applied on this domain before. In the previous works in this field, opponent models have not enjoyed much research interest. The current study not only considers the presence of opponents in these elemental tasks, but also models them as ideal opponents. Thus the opponents are always assumed to act optimally.

The current implementation progresses in two steps. The first step is where the agents learn the action in presence of optimal opponents. This idea, inspired by mini-max criteria in game theory however cannot take advantage of non-optimal behaviours of opponents. Thus in the second phase of the learning task the agents can adapt online to a particular opponent as the play commences. Thus trying implicitly to span from a mini-max strategy to recursive, agent modelled strategies. The second part requires a complete agent to play against an existing opponent. In the absence of a complete agent the second part could not be implemented in practice and is established only as a methodology.

The contribution of the work includes:

- Introduction of radial basis functions in the domain of robot soccer, leading to comprehensibility of learnt models, as shown in figure (1.2).

- Incorporation of optimal adversaries in the learning tasks.
- Expediting data recall from RBFN, by the development of a new tree based implementation called RBF tree.
- Introduction of an analytically incremental function approximation module enabling guaranteed preservation of past training cases.
- Introduction of online learning in robot soccer domain.

Chapter 2

Learning

The problem faced by a learning agent in a state $\mathcal{S}$ is to choose from a set of permissible actions $\mathcal{A}$ a particular instance so as to maximize,a suitably chosen performance measure. These kind of problems can be modelled as Markov Decision Processes.
MDPs are defined by
a set of states: $\mathcal{S}$
a set of actions: $\mathcal{A}$
a set of transition probabilities $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto P$, $P \in [0-1]$
and a set of expected rewards :
$R(\mathcal{S}_t \times \mathcal{A} \times \mathcal{S}_{t+1}) = E[r_t]$

Where r_t is the reward obtained at time t. The performance is evaluated as $\sum \gamma^t r_t$, where γ is a discount rate. The policy $\pi(\mathcal{S}) : \mathcal{S} \times \mathcal{A} \mapsto P$, $P \in [0-1]$ is chosen which maximizes the performance measure. Knowing the model parameters it is possible to find the optimal $\pi^*(\mathcal{S})$ by value iteration or policy iteration [8].

In the event where the model parameters are unknown as is the case for most real world scenarios, different learning techniques can be used. The two popular methods are supervised learning and reinforcement learning.

2.1 Supervised Learning

If it is possible to obtain, at no exorbitant cost , the state to optimal action mapping

$$\mathcal{S} \mapsto \mathcal{A}^*$$

supervised learning is prescribed. This is because supervised learning gives the required action explicitly in the form

$$\mathcal{A}^* = f(\mathcal{S})$$

which can be easily utilised by the agent. The only drawback is that it presupposes the availability of a set of "*instructive*" examples in the form of state-action pairs. Such a training set spanning an adequate portion of the total search space may be difficult or even impossible to provide, in any event it would require an expert and thus would be hard to implement online.

2.2 Reinforcement Learning

In cases where "*instructive*" feedback is not available, another branch of learning algorithms can be used, which can learn with a less demanding availability of "*evaluative*" feedback from the environment. Thus reinforcement algorithms are said to learn from experience as opposed to supervised learning which learn from examples. Reinforcement learning have an added advantage of being able to keep up with dynamic model parameters because they do not need instructive information and thus can be implemented online.

Reinforcement learning however usually does not support an explicit mapping in the form

$$\mathcal{A}^* = f(\mathcal{S})$$

but produce indirect mappings like

$$\mathcal{S} \times \mathcal{A} \mapsto \mathcal{Q} \, \text{or} \, \mathcal{S} \mapsto \mathcal{V}$$

in which case the agent has to search the action value function $\mathcal{Q}(\mathcal{S}, \mathcal{A})$ on or the state value function[1] $\mathcal{V}(\mathcal{S})$ through a maximization operator $argmax()$ to obtain the required action. When these are represented as lookup tables this search is trivial , but not so in case they are represented by different function approximation techniques like neural-nets.

[1]these quantities are defined later. Refer to equations (2.2) and (2.3).

2.2.1 Single Epoch Reinforcement Learning:

One of the difficult issues that has to be addressed by reinforcement learning is that of temporal credit assignment. This stems from the fact that reward may be obtained as a consequence of an action executed far in the past. For learning to be successful it becomes necessary to associate the current reward with the past action.

Single epoch domains, being free from this problem are simpler. The much studied "*k-armed bandit*" [9] problem is such an instance. The problem involves an agent trying to maximize his earnings from a k-armed bandit gambling machine in a fixed number of attempts. Each of the arms ouputs a reward or a penalty based on a probability unknown to the agent. Thus the agent has to balance exploration with exploitation. We will find that the goal shooting task discussed in section [5] to be a similar problem

The *action value function based* approach characterizes each action $\mathcal{A}_i$ with a value $\mathcal{Q}$ signifying the average expected reward.

The optimal action would thus be to choose the action with the highest action value.

$$\mathcal{A}^* =_{argmax\,\mathcal{A}} \mathcal{Q}(\mathcal{A}_i)$$

However the actual action value $\mathcal{Q}$ is unknown to the agent and he has to base his decision on an estimate $\widehat{\mathcal{Q}}$ which he has to learn. A simple strategy would be to take the average of the observed rewards [8].

$$\widehat{\mathcal{Q}_t} = \frac{\sum_0^t r_i}{t} = \widehat{\mathcal{Q}_{t-1}} + \frac{1}{t}(r_t - \widehat{\mathcal{Q}_{t-1}}) \tag{2.1}$$

Thus updating the previous estimate by the new observation with a gradually reducing step length equal to $\frac{1}{t}$.

2.2.2 Infinite/Multi epoch Reinforcement learning:

The quantity to be optimised for an infinite epoch task is modelled as

$$Return = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \ldots = \sum \gamma^i r_{t+i} \qquad 0 < \gamma < 1$$

Where discount factor γ is a mathematical means of bounding the return, and also a means of modelling the intuitive fact that a reward far in future is worth less than an immediate reward. For memory-less dynamics it is possible to model the task as an MDP as stated earlier in this section.

The same value function based approach can be applied, but now the action value function $\mathcal{Q}$ in addition to the action $\mathcal{A}$ will also depend on the current state $\mathcal{S}$, more over we can also define a state value function $\mathcal{V}$ referred simply as value function. These quantities are defined as

$$\mathcal{V}^{\pi}(\mathcal{S}) = E[\sum \gamma^t r_t | \mathcal{S}_t = \mathcal{S}, \pi = \pi(\mathcal{S})] \tag{2.2}$$

this is the expected total discounted rewards for visiting state $\mathcal{S}$ when following policy π.

$$\mathcal{Q}^{\pi}(\mathcal{S},\mathcal{A}) = E[\sum \gamma^t r_t | \mathcal{S}_t = \mathcal{S}, \mathcal{A}_t = \mathcal{A}, \pi = \pi(\mathcal{S})] \tag{2.3}$$

this is the expected total discounted rewards of choosing action $\mathcal{A}$ in state $\mathcal{S}$ and following policy π.
These state and action value function follows the recursive relations shown below.

$$\mathcal{V}(\mathcal{S}) = \sum_{\mathcal{A}} \pi(\mathcal{S}) . \sum_{\mathcal{S}'} \mathcal{T}(\mathcal{S},\mathcal{A},\mathcal{S}') \left[\mathcal{R}(\mathcal{S},\mathcal{A},\mathcal{S}') + \gamma \mathcal{V}(\mathcal{S}')\right] \tag{2.4}$$

$$\mathcal{Q}(\mathcal{S},\mathcal{A}) = \sum_{\mathcal{S}'} \mathcal{T}(\mathcal{S},\mathcal{A},\mathcal{S}') \left[\mathcal{R}(\mathcal{S},\mathcal{A},\mathcal{S}') + \gamma \mathcal{V}(\mathcal{S}')\right] \tag{2.5}$$

Here $\mathcal{S}$ and $\mathcal{S}'$ refer to $\mathcal{S}_t$ and $\mathcal{S}_{t+1}$ respectively. These equations can be used to iteratively update the estimates the after they have been initialised randomly, forming the key element of value iteration and policy iteration. This is basically iterative solution of a system of linear equations. These methods however require the model dynamics to be known as $\mathcal{T}(\mathcal{S},\mathcal{A},\mathcal{S}')$ and $\mathcal{R}(\mathcal{S},\mathcal{A},\mathcal{S}')$ occur explicitly in the formulations.

So there are two approaches to reinforcement learning. One category being the model based methods like certainty equivalent methods [10], Dyna [11], prioritised sweeping [12], RTDP [13]. These use various methods to estimate $\mathcal{R}(\mathcal{S},\mathcal{A},\mathcal{S}')$ and $\mathcal{T}(\mathcal{S},\mathcal{A},\mathcal{S}')$ and solve for the value functions using the equations (2.4) or (2.5). Model free methods on the other hand can work without the knowledge of system dynamics e.g. Actor Critic Methods [14], Q-learning [15] etc.

Actor Critic Methods consists of two components one learning the value function $\mathcal{V}(\mathcal{S})$ the other learning the policy π. Observing an experience tuple $< \mathcal{S}_t, \mathcal{A}_t, r_t, \mathcal{S}_{t+1} >$ the value function is updated as

$$\mathcal{V}(\mathcal{S}_t) = \mathcal{V}(\mathcal{S}_t) + \alpha[r_t + \gamma\mathcal{V}(\mathcal{S}_{t+1}) - \mathcal{V}(\mathcal{S}_t)] \tag{2.6}$$

This is another version of the equation (2.4) with the summing over transition probabilities removed. If the agent explores the environment visiting each site infinitely often, following the transition probability $\mathcal{T}(\mathcal{S}, \mathcal{A}, \mathcal{S}')$ they will converge to the same value.

A positive value of $r_t + \gamma\mathcal{V}(\mathcal{S}_{t+1}) - \mathcal{V}(\mathcal{S}_t)$ implies a preferred action. Thus depending on this value, the policy π is updated. A possible way being:

$$\pi(\mathcal{S}_t) =_{argmax\,\mathcal{A}} \mathcal{Q}(\mathcal{S}_t, \mathcal{A})$$

$$\mathcal{Q}(\mathcal{S}_t, \mathcal{A}) = \mathcal{Q}(\mathcal{S}_t, \mathcal{A}) + \beta[r_t + \gamma\mathcal{V}(\mathcal{S}_{t+1}) - \mathcal{V}(\mathcal{S}_t)]$$

Q-learning on the other hand do not need to learn the value function and works solely on the basis of action values. The update equation for which is given as

$$\mathcal{Q}(\mathcal{S}_t, \mathcal{A}_t) = \mathcal{Q}(\mathcal{S}_t, \mathcal{A}_t) + \alpha[r_t + \gamma_{max\,\mathcal{A}}\mathcal{Q}(\mathcal{S}_{t+1}, \mathcal{A}) - \mathcal{Q}(\mathcal{S}_t, \mathcal{A}_t)] \tag{2.7}$$

2.2.3 Using Function Approximators:

All the algorithms discussed so far assumed that we can identify each state separately and update the value functions accordingly. Proofs of convergence of the above methods exist for such assumptions [16].

However practical implementations, especially for huge and continuous valued states spaces like robot soccer become impossible with the table lookup methodologies, owing to the enormous memory requirement. Possible avenues involve

- partitioning the state space into aggregated blocks, or
- using function approximators.

Function approximators having a long history of wide and in depth study are the most tempting alternative. Both of these approaches however do the same thing that is generalise across the state space. Crisp partitioning of the input state explicitly considers some distance value within which the state value functions are assumed to remain constant. On the other hand for function approximators this fixing of the distance measure is generally not explicit.

Convergence proofs refuse to remain valid when function approximation techniques are used. Pathological cases have been put forward where value function approximation errors have grown arbitrarily large even for simple toy problems [17]. Though convergence is not guaranteed, there has been successful demonstrations also [18]. The main difficulty arises from being unable to control distance over which generalisation takes place. Neural networks are specially susceptible to this drawback as their parameters have nonlocal effect on the learning process. In spite of the absence of convergence proofs the current work will involve Radial basis function networks as a function approximating module.

Domains with large state spaces but with small discrete set of actions, can use separate function approximators for separate actions, or a function approximators having separate output for the separate actions. In all these cases the input to the function learner is the state only. However if the action itself is continuous valued or if the action set is very large the above procedure becomes impractical. In such cases it becomes necessary to include the action also as an input. The function then has a form of $\mathcal{S} \times \mathcal{A} \mapsto \mathcal{Q}$.

Selecting the action then becomes a search executed on this learnt function.Given the current state $\mathcal{S}_t$, the action with the highest value of $\mathcal{Q}(\mathcal{S}, \mathcal{A})$ to find the action with the highest action value given a state. For function learning methods this involves non-trivial computation as opposed to cases where separate action have separate function learners. The standard procedure is to do a gradient ascent on the learnt function. Bairds [19] wire-fitting function approximator gives an example where this search can be greatly simplified.

Here an attempt will be made to execute the primal phase as a supervised learning process by approximating the $\mathcal{S} \mapsto \mathcal{A}$ by radial basis functions. The form $\mathcal{S} \times \mathcal{A} \mapsto \mathcal{Q}$ will also be learned and will be kept relevant in a non-stationary domain with the help of online evaluative feedback , a task difficult if $\mathcal{S} \mapsto \mathcal{A}$ is the only representation that is maintained. The necessary examples or the training set will be generated with the help of multiple simulations.

Chapter 3

Function approximation with Radial Basis Function networks

Radial basis function networks are a class of universal function approximators [20] which have strong resemblance to distance weighted regression or memory based approximations to functions. Unlike sigmoidal feed forward networks, radial basis function networks consists of a number of locally approximating units spanning a suitable amount of the input space. The output of these local units are combined to give the final approximation.

The advantages over feed-forward sigmoidal network includes that the network can be interpreted as a fuzzy logic rule based system thereby offering a lot of comprehensibility; but perhaps the greatest advantage lies in the ease with which it can be trained and the fact that they have only a single minimum which is obviously global[1]. Radial Basis function networks have a closed form solution for the trained weights thereby eliminating time consuming iterative methods. Moreover these units try to approximate the function locally thus any change in the network will affect only a small part of the input space - this makes it possible to edit it incrementally. Moreover these models have theoretical formulations for incremental operation, where new data can be added without disturbing past learnt experiences.

[1]this is for the case where only the weights are considered as modifiable parameters.

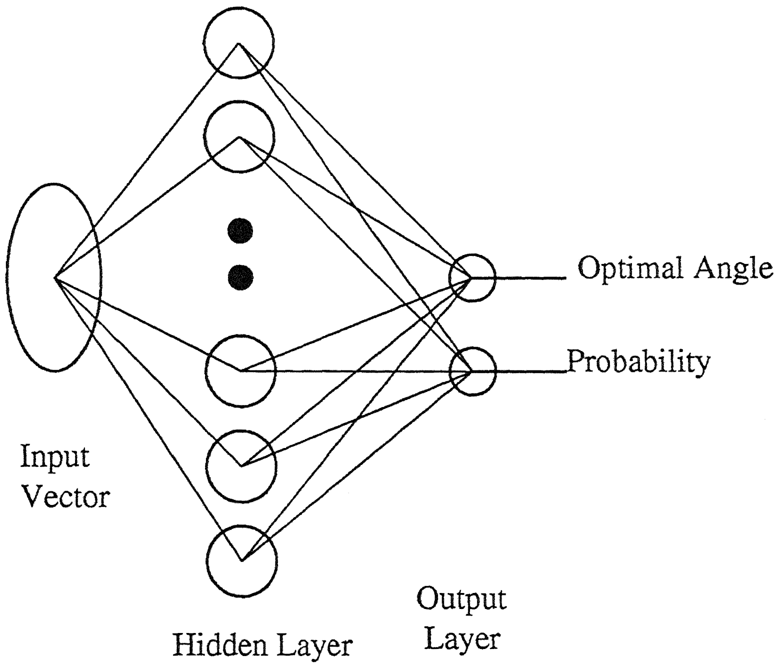


Figure 3.1: Radial Basis Function Network

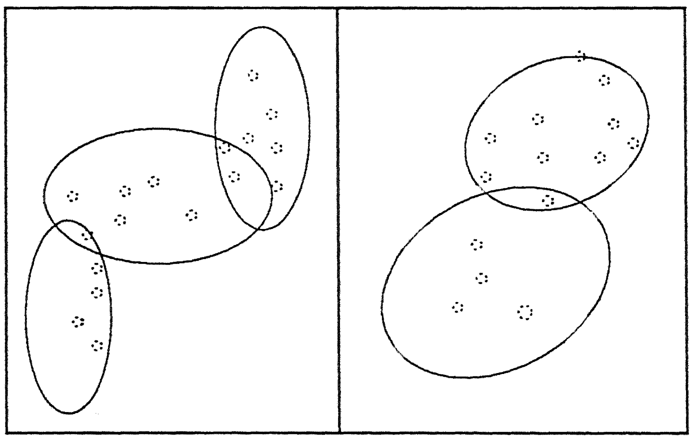


Figure 3.2: Separation into ellipsoidal regions

3.1 Linear combination of Radial basis vectors

A function of the form $f(\overline{x}, \overline{c})$ where $\overline{x}$ is the dependent variable and $\overline{c}$ is a constant vector, is called a Radial Basis Function when it depends only on some distance metric between the vector $\overline{x}$ and the centre $\overline{c}$. The most common form of distance metric used is

$$r = [\overline{x} - \overline{c}]^T [\Sigma]^{-1} [\overline{x} - \overline{c}] \tag{3.1}$$

here the width matrix $[\Sigma]^{-1}$is generally taken to be diagonal. This results in the separation of the input space into a number of hyper ellipsoids as shown in fig [3.2]. Depending on whether the $[\Sigma]^{-1}$ is diagonal or not the major axis of these ellipsoids are aligned or at an angle with the axis of the input vectors. Non-diagonal matrices naturally offers more flexibility but at the cost of more parameters to tune. This can be conceived as a 4 layer network, i.e. having an extra linear preprocessing layer such that it output is given as

$$X = [W].[\overline{x} - \overline{c}]$$

Thus the new metric now gives

$$r = [X]^T [\Sigma]^{-1} [X]$$

$$r = [\overline{x} - \overline{c}]^T \left[W^T \Sigma^{-1} W \right] [\overline{x} - \overline{c}]$$

Many forms of the function have been used - multi-quadric, inverse multi-quadric, thin plate splines but the most popular is the Gaussian form.

$$f(r) = \frac{1}{\sqrt{2\pi}} e^{-\frac{r^2}{2\sigma^2}}$$

An RBF consists of a number of such radial basis function units, whose outputs are combined linearly to give the final output. Thus the output is given as

$$\hat{y} = \sum_{1}^{n} w_i f_i(r_i) \tag{3.2}$$

Given a training set in the form of N dataset consisting of the input vector $\overline{x_i} \in \mathcal{R}^d$and the desired output y_i we find the values of w_i which minimizes the error

$$E = \sum_{1}^{N} (y - \hat{y})^2 \tag{3.3}$$

Considering the centres $\overline{c_i}$ to be known this minimization is linear in the parameters to be determined. The output of the RBF net can be written as

$$\begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} f_{11} & \dots & f_{n1} \\ \vdots & \ddots & \vdots \\ f_{N1} & \dots & f_{Nn} \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$$

or

$$Y = FW$$

Thus essentially the radial basis networks convert a $\mathcal{R}^d$ feature vector to a vector $\mathcal{R}^n$usually of higher dimension; which are linearly combined in the ratio given by the vector $\overline{w}$ to give the necessary approximation.

The closed form expression of the weights $\overline{w}$ can be obtained by elementary linear algebra as

$$\overline{w} = \left(F^T F\right)^{-1} F^T Y \tag{3.4}$$

This solution is unique and minimizes the network globally.
It is also possible to implement a normalized form of RBF nets where

$$\hat{y} = \sum_{1}^{n} w_i \frac{f_i}{\sum_1^n f_i} = \frac{\sum_1^n f_i.w_i}{\sum_1^n f_i} \tag{3.5}$$

This form is utilized as this can be interpreted as a fuzzy inference engine as well as in a Bayesian framework as will be shown shortly.

3.2 Different error measures

3.2.1 Ridge regression:

The error criteria considered for minimization in equation (3.3) does not capture all the qualities desired for a good generalizing network. One such requirement is that the weight terms do not shoot up, which usually results in over-fitting of data and roughness of the learnt surface. This criteria can be encoded in the form of a penalty term

$$E = \sum_{1}^{N}(y-\hat{y})^2 + \sum_{1}^{n}\lambda_i w_i^2$$

$$E = [y-\hat{y}]^T[y-\hat{y}] + [\lambda]^T[w]^T[w] \qquad (3.6)$$

The modified weights are given by

$$W = [F^T F + diag[\lambda]\,]^{-1}F^T[Y]$$

The factors λ_i are called the ridge parameters and helps in improving the condition number of $[f_{ij}]^T[f_{ij}]$ thereby making the inversion more stable.

3.2.2 Weighted error criteria:

The error function (3.3) gives equal importance to all the training cases. Sometimes it is necessary to weight the instances differently. Consider a typical case of learning the desired action $\mathcal{A}$ which maximizes the action value $\mathcal{Q}(\mathcal{A}_i)$.

$$_{argmax\,\mathcal{A}_i}\,\mathcal{Q}(\mathcal{A}_i) = \mathcal{A}^*$$

as

$$\widehat{\mathcal{A}^*} = \sum f_j w_j$$

The previous error term (3.3) will penalise the difference between the optimum action value $A^*$and the predicted action value. However we should be more concerned with the loss in the actual performance as a result of the error in action selection i.e.

$$e = \mathcal{Q}(\mathcal{A}^*) - \mathcal{Q}(\widehat{\mathcal{A}^*})$$

.

By expanding about the optimum action value we get

$$e = \mathcal{Q}(\mathcal{A}^*) - \mathcal{Q}(\mathcal{A}^*) - \frac{\partial \mathcal{Q}}{\partial \mathcal{A}}\,|_{\mathcal{A}=\mathcal{A}^*}\,\Delta\mathcal{A} - \frac{\partial^2 \mathcal{Q}}{\partial \mathcal{A}^2}|_{\mathcal{A}=\mathcal{A}^*}\Delta\mathcal{A}^2 + \ldots$$

$$e = -\frac{\partial^2 \mathcal{Q}}{\partial \mathcal{A}^2}\,|_{\mathcal{A}=\mathcal{A}^*}\,\Delta\mathcal{A}^2 + \ldots$$

We want to minimize the total magnitude of this error over the training set $1 \leq k \leq N$. We note that $\Delta\mathcal{A}^2$ is always positive thus it is enough to minimize

$$\min \sum_{k=1}^{k=N} abs(\frac{\partial^2 \mathcal{Q}_k}{\partial \mathcal{A}^2} \mid_{\mathcal{A}=\mathcal{A}_k^*}) \Delta\mathcal{A}^2$$

considering $abs(\frac{\partial^2 \mathcal{Q}_k}{\partial \mathcal{A}^2} \mid_{\mathcal{A}=\mathcal{A}_k^*}) = s_k$ we get the modified error term as

$$E = [\mathcal{A}_k^* - \widehat{\mathcal{A}_k}]^T S [\mathcal{A}_k^* - \widehat{\mathcal{A}_k}]$$

where S is $diag(s_k)$ The solution of weights for which can be found to be

$$W = [F^T S F]^{-1} F S A$$

3.2.3 Cross-validation

In addition to minimizing a suitable error criteria over the training set it is of vital importance to check how the learnt function approximator generalizes to cases not present in the training set. The method of estimation of generalisation error is to divide the available data into a training and a validation set. The function is trained using data only from the training set and then tested over the validation cases.

Therefore it becomes necessary to consider, how to partition the available data into two disjoint sets. Moreover keeping data for validation implies a missed opportunity for training . When obtaining data is costly this can be a significant loss.

A procedure to deal with this is to use *leave one out cross validation* (LOO). Given a training instance of N cases it is split into N-1 training cases and one testing case. This partition can be done in N different ways. LOO cross-validation is the average of these testing error.

Radial Basis Networks being linear models enjoy a closed form solution of the LOO cross validation in terms of the projection matrix P.

$$\sigma_{LOO}^2 = \frac{Y^T P^T (diag(P))^{-2} P Y}{N}$$

Where the projection matrix P is related to the error vector as

$$E = [Y - \hat{Y}] = Y - FW = Y - FA^{-1}F^TY = [I - FA^{-1}F^T] = PY$$

Where $A = F^TF$

A common simplification to σ^2_{LOO} is to replace the diagonal matrix $diag(P)$ with another diagonal matrix having all the elements equal to the average of those in $diag(P)$. This simplified criteria is called generalized cross-validation and given as

$$\sigma^2_{GCV} = \frac{Y^TP^TIPY}{N(\frac{trace(P)}{N})^2} = \frac{N\sigma^2_{MSE}}{(trace(P))^2}$$

3.3 Online adaptation of RBF weight parameters

All the methods presented so far assumed that the entire training data was available beforehand, thus these are offline methods. However for reinforcement learning one has to adapt online to new experience $(y_{new}, \overline{x_{new}})$.

The most common method is to use line search on the error gradient direction $-\nabla E$. This can however destroy previous learning instances already encoded into the parameter values w. To prevent this one usually takes a small step along the gradient instead of a full line search.

$$\Delta w = -\alpha \nabla E|_{x=x_{new}} \quad \alpha = small$$

This however cannot utilize the new data sufficiently. Thus it is necessary to have an incremental update rule that can preserve all the previous instances. Radial basis function Networks is one of few smooth function approximators for which this can be derived.

Most of the update rules of reinforcement learning follow the pattern where observation (y_{new}, x_{new}) is used to form the training set $(y_{predicted}, x_{new})$ which is used to update the network as

$$\widehat{y_t} = \widehat{y_{t-1}} + \alpha(t)[y_{predicted}(y_{new}) - \widehat{y_t}]$$

If the new training instance happens to coincide with a previous instance (y_{old}, x_{old}) one can simply replace the y_{old} by $y_{predicted}$, and obtain the weights by making changes only in the Y matrix. Otherwise we have to induct it as new training instance. This will add another row to the matrix F giving the new matrix as

$$F_{new} = \begin{bmatrix} F \\ \cdots \\ \Phi \end{bmatrix}$$

where $\Phi_{ij} = f_i(\overline{x_{new}}, \overline{c_j})$. The new weights for which can be given as

$$W_{new} = [F_{new}^T F_{new}]^{-1} F_{new}^T \begin{bmatrix} Y_{old} \\ \cdots \\ y_{predicted} \end{bmatrix}$$

$$W_{new} = \left[F_{old}^T F_{old} + \Phi^T \Phi\right]^{-1} \left[F_{old}^T Y + \Phi^T y_{new}\right]$$

In this case the value of $F_{old}^T Y$ will already be known and the matrix inverse on the left can be expressed in terms of the pre-calculated inverse matrix $[F_{old}^T F]^{-1}$ using the matrix inversion lemma [21] as

$$[A + \Phi^T \Phi]^{-1} = A^{-1} - \frac{A^{-1}\Phi^T \Phi A^{-1}}{1 + \Phi A^{-1} \Phi^T}$$

This saves a considerable amount of computation especially through the avoidance of matrix inversion thereby reducing the complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$.

The RBF model is analytically incremental, not only in terms of addition of new data but also in terms of addition or deletion of centres. This offers a great advantage by allowing induction of new centres without the need for re-tuning from the beginning.

The addition of a new centre will add a new column to the F matrix

$$F_{new} = \begin{bmatrix} F_{old} & \vdots & \Phi \end{bmatrix}$$

keeping the Y matrix unchanged.

The changed weights will be given by

$$W_{new} = \left(F_{new}^T F_{new}\right)^{-1} F_{new}^T Y$$

or

$$W_{new} = \begin{bmatrix} F_{old}^T F_{old} & F_{old}^T \Phi \\ \Phi^T F & \Phi^T \Phi \end{bmatrix}^{-1} \begin{bmatrix} F_{old} & \vdots & \Phi \end{bmatrix}$$

The inverse of the matrix on the left can be evaluated utilizing its block partitioned structure. Note that $F_{old}^T\Phi$ and $\Phi^T F$ are column and row vectors and $\Phi^T\Phi$ is a scalar. The inverse of a matrix having the following structure is given as

$$\begin{bmatrix} A & v_1 \\ v_2^T & s \end{bmatrix}^{-1} = \begin{bmatrix} X & -\frac{Xv_1}{s} \\ \frac{v_2^T X}{s} & w \end{bmatrix}$$

where v_1 and $v_2$are vectors and s is a scalar. The other terms are given as

$$X = [A - \frac{v_1 v_2^T}{s}]^{-1} = A^{-1} + \frac{A^{-1}(v_1 v_2^T)A^{-1}}{1 + v_2^T A^{-1} v_1}$$

no new matrix inversion is necessary as A^{-1} is already pre-calculated.

$$w = \frac{1}{s}(1 + \frac{v_2^T X v_1}{s})$$

In this case also computational complexity is of the order $\mathcal{O}(n^2)$.

3.4 Fuzzy reasoning

Fuzzy logic is a popular means of approximating complex functions or models using linguistic rules. They have been used extensively in complicated control problems [22]. Jang [23] proposed a generalised neural network representation of a fuzzy inference system, thereby allowing tuning of the fuzzy model by backpropagation. Optimising a fuzzy system is an area of active research and various methods based on gradient descent as in [24] and genetic algorithms [25] have been used.

A fuzzy inference system can be defined for a d dimensional input of the form $\overline{x} \in \mathcal{R}^d$ and an output y by a set of n rules. Each rule is defined as

$$R_i : \text{if } \mathrm{x_1} \text{ is } \mathrm{A_{i1}} \text{ and } \mathrm{x_2} \text{ is } \mathrm{A_{i2}} \ldots \mathrm{x_d} \text{is } \mathrm{A_{id}} \text{ then y is } \mathrm{C_i}$$

where A_{ij} forms the antecedent associated with the i^{th} rule and the j^{th} component of the input vector and C_i forms the consequent for the i^{th}rule.

For each of the antecedents membership functions

$$\mu_{A_{ij}}(x_j) \mapsto \mathcal{R}$$

map the variable values to a scalar signifying the degree of membership of the variable to the fuzzy antecedent.

The strength of firing of the i^{th} rule consisting of a conjunction of such antecedents is interpreted depending on how conjunction is defined in the inference engine. Two common method are the *min* operator and the *product* operator. Using the *product* operator we get the strength of the rule as

$$\mu(\overline{x})_i = \prod_1^d \mu_{A_{ij}}$$

Depending upon the implementation the consequent may be modelled as a fuzzy set with its own membership functions or as in [26] by constants. In the latter case the final output of the fuzzy inference engine is given as

$$y = \frac{\sum_{i=1}^{i=n} \mu_i(\overline{x}) C_i}{\sum_{i=1}^{i=n} \mu_i(\overline{x})} \tag{3.7}$$

or if the memberships are already normalized as

$$y = \sum_1^n \mu_i(\overline{x}) C_i$$

The shape of the membership functions is free to be implemented as desired. The most simple being triangular and trapezoidal membership functions. For modelling highly nonlinear phenomena Gaussian membership functions are popular . A Gaussian membership function is given as

$$\mu_i = k \exp^{-1/2\left(\sum \frac{(x_{ij}-c_j)^2}{\sigma^2}\right)}$$

$$\mu_i = k \exp^{-1/2([x-c]^T[\Sigma]^{-1}[x-c])} \tag{3.8}$$

Comparing the equation (3.8) and equation (3.7) with the equations (3.2) (3.1) we see that the output of the each radial basis unit can be read as the strength of firing of a rule and the consequents can be interpreted as the weights. This conveys another advantage to the RBF networks over the more common sigmoidal feed-forward neural networks, by the fact that they can be interpreted in terms of linguistic rules.

3.5 Clustering as Antecedent induction

As mentioned in section (3.1) radial basis network parameters consists of a centre $\bar{c}$ which is determined a priori and a weight $\overline{w}$ which is found by solving the resulting linear least square problem. From section (3.4) we see that the centre corresponds to the antecedent portion of the fuzzy inference engine whereas the weights correspond to the consequents.

Different clustering techniques which are used for finding the centres, in effect generate the antecedents of the rule and can be seen as a rule induction method. However there is a difference how the rules are induced for fuzzy inference engines and radial basis functions. In fuzzy rule based systems the universe of discourse of a particular variable is split into some number of linguistic sets to form the antecedents. These are then combined exhaustively with the antecedents of other variables resulting in a rule base size exponential in the the number of linguistic partitions.

For radial basis function networks, it is generally pre-decided how many rules or centres are to be used. The entire data set is divided into as many clusters and the centres of those are taken to be the the antecedents. This automatically allocates the location of the membership function according to the distribution of the data. This leads to finer partition of regions rich in data. If the number of rules or centres prove insufficient more can be added in the regions of poor approximation.

RBFs however are badly effected by the presence of variables which are irrelevant. As in that case it has to take into account the entire space of the variable combined with all other antecedents so as to learn to ignore the effects of that particular variable.

3.5.1 Hard k-means clustering

Clustering can be visualised as a memory based learning or data compression where the dataset consisting of p instances of a d dimensional vector $x \in \mathcal{R}^d$ is partitioned into $1 \leq k \leq p$ clusters, each of which has a centre in the form of a vector c_j that best resembles all the vectors allocated to the cluster j. The degree of membership of an instance x to a cluster j , is represented by the membership function

$$\mu_j(x)$$

In hard clustering each instance of the training set can be allocated to at most a single cluster, or in other words the membership function is constrained to be 1 or 0.

The *hard k means clustering* is started with a set of estimated centre locations $\hat{c}_{i,t=0} \in \mathcal{R}^d$. Following which the following steps are executed.

1. calculate the membership of the k^{th} instance to the j^{th}centre

$$\mu_{j,k} = \begin{cases} 1 & if\, \forall l \neq j \; \|x_k - c_j\|^2 < \|x_k - c_l\|^2 \\ 0 & otherwise \end{cases}$$

2. calculate the new estimates as

$$c_{j,t+1} = \frac{\sum_k \mu_{j,k} x_k}{\sum_k^k \mu_{j,k}}$$

3. calculate shift as

$$E = \sum_1^n \|c_{j,t+1} - c_{j,t}\|^2$$

4. if shift $E < \epsilon$ then exit else repeat from 1

The procedure of hard k-means clustering is guaranteed to converge, but may get trapped in local minima. To alleviate this problem we add a gradually decreasing noise level to the centre update equation. The annealing schedule followed was

$$\mathcal{N} = \mathcal{N}_i \left(\frac{\mathcal{N}_f}{\mathcal{N}_i}\right)^{\frac{iteration}{Max\,iteration}}$$

where $\mathcal{N}_i$ and $\mathcal{N}_f$ refer to the noise levels at the initial and final stages.

3.5.2 Soft k-means clustering

In soft clustering an instance of the training set can belong to different clusters with different degrees of membership. The membership values $\mu < 1$ but need not sum to 1. Different membership functions can be formed based on the Euclidean distance $d_{kj} = \|x_k - c_j\|^2$ of the instance from the centre. A popular implementation is as follows [27].

1. set initial estimates of the centres $c_{j,t=0} \in \mathcal{R}^d$

2. calculate the membership values, for a softness index m

$$\mu_{j,k} = \frac{1}{\sum_l (\frac{d_{k,j}}{d_{k,l}})^{\frac{1}{(m-1)}}}$$

3. calculate the new estimates as

$$c_j = \frac{\sum_j (\mu_{j,k})^m x_k}{\sum_j (\mu_{j,k})^m}$$

4. calculate error

$$E = \sum_1^n \|c_{j,t+1} - c_{j,t}\|^2$$

5. if error $E < \epsilon$ then exit, else repeat from 2

here as $m \to 1$ the algorithm approaches the hard-k-means algorithm and as $m \to \infty$ the more soft is the clustering.

3.5.3 Expectation Maximization clustering

Expectation Maximization [28] is an algorithm to find the Bayesian Maximum Likelihood estimate of a parameter (vector in this case) when some information is missing. Its typical use is in the estimation of the means and variances of a mixture model from the raw data given the number of mixtures.

Let us assume that we have a mixture of n Gaussian processes each characterized by their mean vector c_j and the covariance matrix $[\Sigma_j]$. The data set is generated 1st by choosing a particular Gaussian process j following which a data set is generated using the given probability density function.

The objective is to determine the maximum likelihood estimate of the parameters c_j, $[\Sigma_j]$ given a data set x_k $k = 1 \ldots N$ without access to the information about which process j was used to produce the data x_k.

This is solved using a Bayesian analysis as follows

let prior probability that the process j is selected be given by

$$p_{centre_j} = 1/n \ldots initially$$

The probability that the data x_k came from the process j can be inferred by Bayes theorem as

$$P(centre = j/x_k) = \frac{p(x_k/centre = j).p_{centre_j}}{\sum_{j=1}^{n} p(x_k/centre = j).p_{centre_j}} \tag{3.9}$$

The probability $p(x_k/centre = j)$ is known given the parameters μ and $[\Sigma]$. This is given as

$$p(x_k/centre = j) = (2\pi)^{-\frac{d}{2}} Det(\Sigma_j^{-1}) e^{-\frac{1}{2}[(x_k - c_j)^T [\Sigma_j]^{-1} (x_k - c_j)]}$$

for such a model the expectation of y given the value of x is given as

$$E[y/x] = \frac{\sum_j p(x_k/centre = j).y_j}{\sum_j p(x_k/centre = j)}$$

The objective is to maximize the likelihood

$$\prod_{k=1}^{k=p} P(centre = j/x_k)$$

with respect to the parameters c_j and $[\Sigma]$. The maximization is simplified by maximizing the log of this likelihood as

$$\max \sum_{k=1}^{k=p} \log P(centre = j/x_k)$$

The algorithm can be defined in terms of 2 steps

1. Estimation: which involves calculating the probability of ownership given the current estimate of the parameters $c_{j,t}$ $[\Sigma_{j,t}]$. This is obtained from equation (3.9)

2. Maximization: which involves finding the new values of the parameters c_j and $[\Sigma_j]$ which maximizes the log-likelihood given the current estimate of the probability of ownership. This step reduces to weighted averaging process as shown

$$c_{j,t+1} = \frac{\sum_k P(centre = j/x_k).x_{k,t}}{\sum_k P(centre = j/x_k)} \tag{3.10}$$

The elements of the matrix $[\Sigma_j]$ are also similarly obtained. If ${}_{r,c}\Sigma_j$ denotes the element at the r^{th} row and c^{th} column of the matrix it can be evaluated as

$$_{r,c}\Sigma_{j,t+1} = \frac{\sum_k P(centre = j/x_k).{}_{r,c}\Sigma_{j,t}}{\sum_k P(centre = j/x_k)}$$

In case the covariance matrix is diagonal one can directly obtain the the terms of $[\Sigma_j]^{-1}_{t+1}$.

3.5.4 Gradient Descent Clustering:

The centre parameters $\bar{c}$ and σ can also be obtained by operating a gradient descent on the error function. This is facilitated by the fact that the RBF form is analytically differentiable.

Using the error function (3.3) the gradient with respect to a parameters of the k^{th} centres θ_k is obtained as

$$\frac{\partial E}{\partial \theta_k} = -\sum 2(y - \hat{y})\frac{\partial \hat{y}}{\partial \theta_k} \tag{3.11}$$

from the equation (3.5) we obtain

$$\frac{\partial \hat{y}}{\partial \theta_k} = w_k \frac{(\sum_1^n f_i)\frac{\partial f_k}{\partial \theta_k} - f_k \frac{\partial}{\partial \theta_k}(\sum_1^n f_i)}{(\sum_1^n f_i)^2} - \sum_{j \neq k} \frac{w_j f_j \frac{\partial}{\partial \theta_k}(\sum_1^n f_i)}{(\sum_1^n f_i)^2}$$

or

$$= \frac{(w_k(\sum_1^n f_i) - \sum_1^n w_i f_i)\frac{\partial f_k}{\partial \theta_k}}{(\sum_1^n f_i)^2}$$

$$= \frac{(w_k - \hat{y})}{\sum_1^n f_i}\frac{\partial f_k}{\partial \theta_k} \tag{3.12}$$

For updating the k^{th} centre parameters $\overline{c_k}$ and $\overline{\sigma_k}$ we use appropriate terms in place of θ. These are given by

$$\frac{\partial f_k}{\partial c_{ki}} = \frac{\partial}{\partial c_{ki}} \left(e^{-\frac{1}{2}\sum_i \frac{(x_{ki}-c_{ki})^2}{\sigma_{ki}^2}} \right) = f_k \frac{(x_{ki} - c_{ki})}{\sigma_i^2} \tag{3.13}$$

and

$$\frac{\partial f_k}{\partial \sigma_{ki}} = \frac{\partial}{\partial \sigma_{ki}} \left(e^{-\frac{1}{2}\sum_i \frac{(x_{ki}-c_{ki})^2}{\sigma_{ki}^2}} \right) = f_k \frac{(x_{ki} - c_{ki})^2}{\sigma_i^3} \tag{3.14}$$

These equations can be chained to give the necessary update rule in the form

$$\theta_{t+1} = \theta_t - \alpha \nabla E$$

Since this procedure directly minimizes the error the final result obtained by this is obviously better, but this method is more computationally intensive than the others. Good results are obtained by interlacing this centre update rule with recalculating the weights w by the equation (3.4).

Chapter 4

Tree Based Representation

In addition to the time taken for training the system, one also has to consider the ease of retrieval. Lazy learning techniques for instance, defer all processing till a prediction is to be made. Thus they have practically zero training expense, but retrieval costs are substantially higher, especially with large training sets.

For the case of radial basis function networks with a large number of centres, often only a few of the centres have a significant contribution. A methodology which can eliminate centres which are insignificant for the current input, will save a lot of computation and will make the retrieval faster.

Methods for accelerating memory-based learning had been proposed by Deng and Moore [29] which stores the instances as a *k-d tree*, thereby helping in reducing the computational cost of finding the nearest neighbours.

The current implementation of radial basis function tree will try to partition the input space recursively, such that within a partition only a handful of the centres are active. Thing to be noted is that the referred implementation [29] partitions the training instances while this will partition the input space.

4.1 K-d Tree assisted Memory based learning:

A *k-d tree* is a special binary tree data structure used for storing multiple instances of a k dimensional vector $\overline{x} \in R^k$ in a systematic manner. Each non-terminal node having a right and a left child, splits the exemplar set into 2 disjoint sets on the basis of a particular component of the vector $\overline{x}$. The

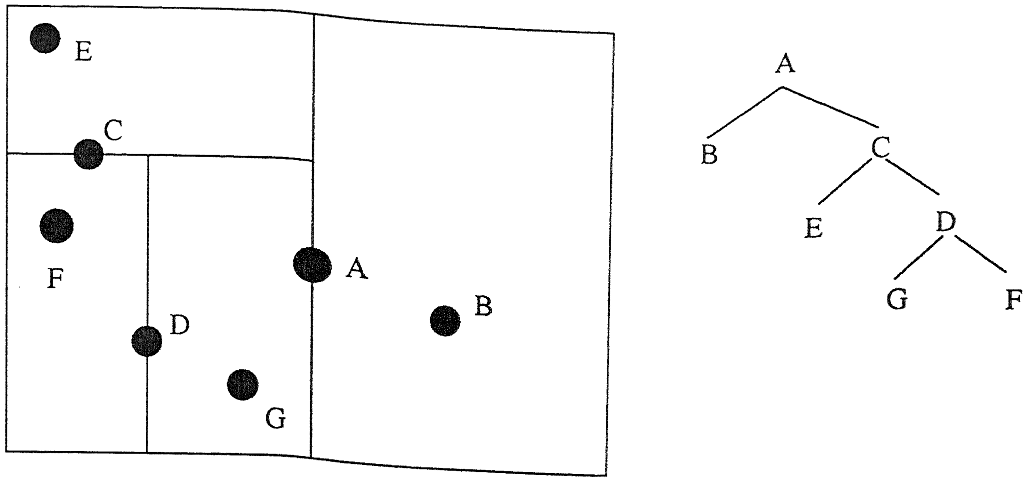


Figure 4.1: K-d tree partitioning of a 2d space

component used for splitting is called the splitting variable and the value used for splitting is the splitting value. All instances of the exemplar set having the value of the splitting variable higher than the splitting value chosen are located in the left sub-tree and the others on the right sub-tree. *K-d trees* are popular for accelerating multi-dimensional search as binary search trees are popular for single dimensions [30]. A *k-d tree* partitioning of a 2d space is shown in fig [4.1].

Formally let $\mathcal{E}$ signify the exemplar set of instances of $\overline{x_i}$ with i ranging from 1 to N. Then the root node of the tree T is defined as a set

$$T_{root}\{x|\, x \in E\}$$

$$T_{root} = T_{left\,child} \cap T_{right\,child}$$

$$T_{left\,child}\{x|\,(x \in E) \cap (x_i > split_i)\}$$

$$T_{right\,child}\{x|\,(x \in E) \cap (x_i \leq split_i)\}$$

4.1.1 Kernel Regression:

Kernel regression is a special case of memory based learning very similar to that of radial basis function networks. In this case all the past training data

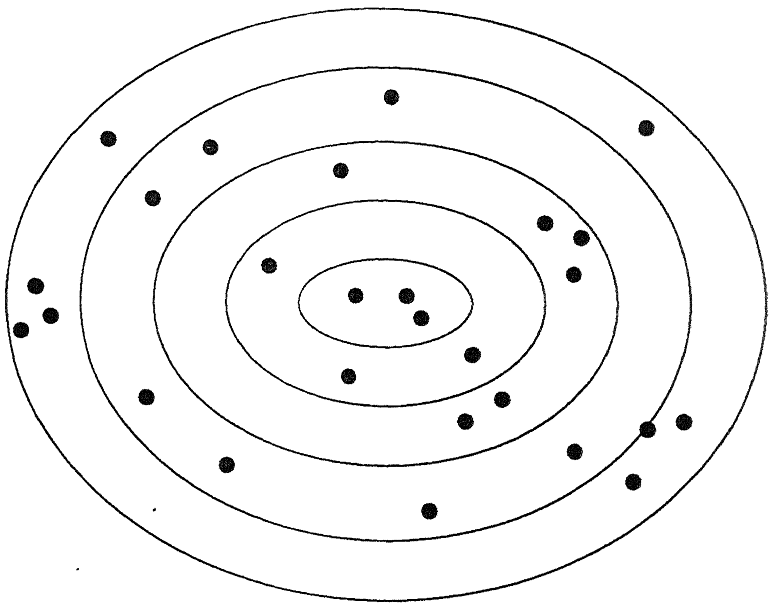


Figure 4.2: iso-response elliptical bands superposed over distribution of data

N are used to produce the output for the new query point $\overline{x_q}$. The output is given as

$$y(\overline{x_q}) = \frac{\sum_{i=1}^{i=N} K(dist_i(\overline{x_q}))y_i}{\sum_{i=1}^{i=N} K_i(dist_i(\overline{x_q}))} \tag{4.1}$$

Where $K(dist_i(\overline{x_q}))$ is the kernel function depending on the distance metric $dist_i(\overline{x_q})$ of $\overline{x_q}$ from the training instances $\overline{x_i}$. Comparing this with radial basis function networks we see that this is same as an RBF with all the past training cases acting as centres and using the same distance metric for all the cases and using the Gaussian as the kernel function.

Thus for hyper-ellipsoidal bands of narrow thickness defined over the training instances as shown in [4.2] , the value of the kernel function would not vary much.The kernel functions can then be approximated by the kernel function value at the centre of this band.Thus if we divide it into l number of such bands of almost constant kernel values the regression can be obtained as

$$y(\overline{x_q}) = \frac{\sum_{i=1}^{i=l} n_i.K(dist_{mid_i}(\overline{x_q}))y_i}{\sum_{i=1}^{i=l} n_i.K_i(dist_{mid_i}(\overline{x_q}))} \tag{4.2}$$

note now we have to sum over $l << N$, n_i is the number of training cases within the band, $dist_{mid_i}(\overline{x_q})$ is the kernel value at the centre of the band i.

4.2 Radial Basis Function trees:

Deng and Moore proposed storing every instance of the training set on a k- d tree. Given a query point it was possible to identify the minimum and maximum response of the kernel function within each partitioned rectangloid. This information then could be used to decide whether to traverse that branch further. Such a method of pruning the search, cannot be used for accelerating radial basis function networks. This difficulty arises from the fact that each of the centres uses a different distance metric. Had the matrix $[\Sigma]$ used in the distance metric formulations of the centres (equation 3.1) been same for all the centres then the same technique as above could have been applied.

Following this difficulty a different approach is taken as explained. The entire input space is taken to be a finite hyper rectangle and is divided into a number of blocks. The simplest of which would be a grid partitioning. For each of these blocks it is necessary to identify the set of radial basis units which have a significant output at any point within it, these are maintained as a list. A radial basis function unit f_i is considered insignificant if

$$(\forall \overline{x}) \in block \Rightarrow f(\overline{x}) \leq \epsilon$$

$$or \max_{x \in block} f(\overline{x}) \leq \epsilon \tag{4.3}$$

the efficiency of this algorithm hinges on the ease with which we can calculate the expression (4.3), note however that this has to be done only once. For small enough sizes of the blocks the effective number of centres n_{eff} within the block would be small, provided we can easily identify the block given a query point this will sufficiently reduce recall expenses.

4.2.1 Finding maximum Response of a RBF unit within a hyper rectangle:

For the centre c_i of the radial basis unit located within a hyper-rectangle the maximum will be at the centre itself. For centres outside the hyper rectangle the point where the maxima occurs needs to be found.

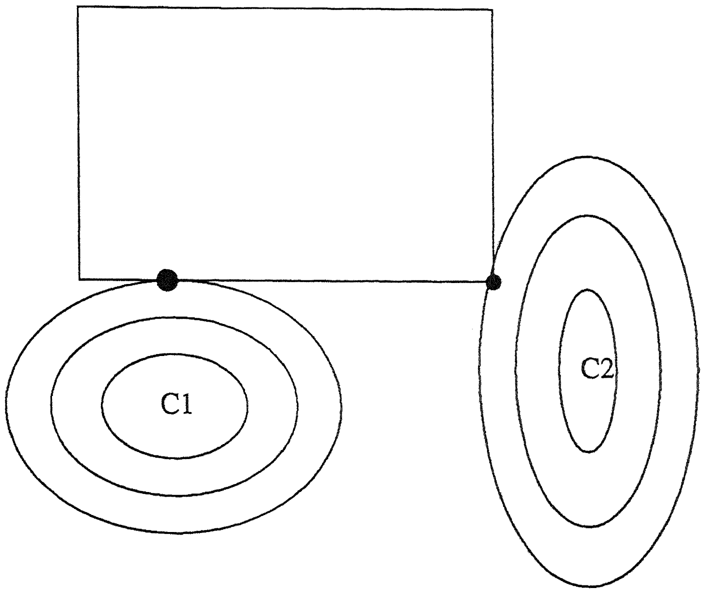


Figure 4.3: Location of the minimum distance norms for a given input space and different radial basis centres.

Since equi-response lines of RBF units are hyper-ellipsoids, the process of finding the point with maximum response reduces to finding the point of intersection of the hyper-rectangle with the smallest of the system of concentric hyper-ellipsoids around the centre c_i.

It can be noted that this will be located only on the boundary of the hyper-rectangle. The point will be located either on a vertex or a hyper-plane forming a face of the hyper -rectangle. Moreover the point will be located on a hyper-plane only when the ellipsoid makes a tangential contact with the hyper-rectangular block.

For axis parallel ellipsoids such tangential contact can occur only along the major and minor axes, thus such points will lie on the intersection of the axes with the boundary of the hyper-rectangle figure [4.3]. For inclined ellipsoids one can identify a system of lines through the centre along which the points of tangency can lie, thus one needs to search only the points of intersection of these lines with the boundary of the hyper-rectangle.

However in the case of axis parallel ellipsoids, as is being used in the current implementation of radial basis units, the identification of the point of maximum response is computationally trivial. Consider that the hyper-rectangle is specified by the upper and lower limits of each dimension L_i and H_i and the components of the centre of the radial basis unit to be given as c_i. Let the point of maximum response be p_i. The components of p are obtained as

$$p_i = \begin{cases} c_i & if \quad L_i \leq c_i \leq H_i \\ H_i & if \quad c_i \geq H_i \\ L_i & if \quad c_i \leq L_i \end{cases}$$

4.2.2 Partitioning:

The previously mentioned strategy of grid decomposition is inefficient as it imposes the constraint of equality on the size of the blocks. Unequal blocks on the other hand are expected to require less number of partitions. The methodology to be followed would be to recursively partition the input space hyper-rectangle along different dimensions till the effective number of centres n_{eff} for the blocks fall below a threshold. The most intuitive representation of such a partitioning is a *k-d tree.*

The objective of the *k-d tree* is to reduce the number of effective centres within each partition. This is done by suitably choosing the splitting dimension and the splitting value. Given a splitting dimension the optimum choice of a value is one which follows the the minimax criteria, that is which minimizes the maximum n_{eff} of the two partitions produced

$$split\, value = \min_d \max n_{\text{eff}}$$

Once the mechanism for selecting the splitting value is specified, we can identify the splitting dimension as that dimension which results in the maximum reduction. The optimum splitting value can be obtained by executing search along each of the dimensions, but that will be computationally intensive.

To obtain the splitting value we adopt two heuristics

1. Median value split: This ensures equal number of blocks in each partition, but the effective number of centres for each block may not be adequately reduced as a result of contribution from centres beyond its boundaries.

2. Select the splitting hyperplane so as to minimise the response of all centres on it. This reduces to selecting that hyperplane which maximises the sum total of the different distance metric used by the centres.

We evaluate both the heuristics, and adopt the better. It has been seen that the latter approach gives better results, more often.
For axis parallel radial basis units the distance metric is of the form

$$\sum_k \frac{(x_k - c_k)^2}{\sigma_k^2}$$

refer to (3.1). The sum of this distance is to be maximised while ensuring that the hyperplane remains within the convex hull of the centres. This constrained optimisation problem

$$\max_{s_{k,i}} \sum_k^{no\,centres\,on\,list} \tfrac{(s_{k,i}-c_{k,i})^2}{\sigma_{k,i}^2} \quad \text{st} \quad s_{k,i} = \textstyle\sum_k w_k c_k \ and \ \sum_k w_k = 1$$

can be formalised with the help of Lagrangian multipliers as

$$\max_{s_{k,i}} \sum_k \frac{1}{\sigma_{k,i}^2}(c_{k,i} - \sum_k w_k c_k)^2 + \lambda(\sum_k w_k - 1)$$

differentiating with respect to w_j we get

$$2x_{j,i}\left(\sum \tfrac{c_{k,i}}{\sigma_{k,i}^2} - \sum w_k c_k . \sum \tfrac{1}{\sigma_{k,i}^2}\right) + \lambda = 0 \quad \forall j$$

$$\left(\sum \tfrac{c_{k,i}}{\sigma_{k,i}^2} - \sum w_k c_k . \sum \tfrac{1}{\sigma_{k,i}^2}\right) = -\tfrac{\lambda}{2x_j} \quad \forall j$$

This is only possible when expressions on both sides of the equality are individually zero. Thus we obtain the splitting value

$$s_i = \sum w_k c_k = \frac{\sum \frac{c_{k,i}}{\sigma_{k,i}^2}}{\sum \frac{1}{\sigma_{k,i}^2}}$$

Thus these two heuristic consists of selecting the median or a weighted mean of the centre values. The splitting variable can either be chosen by evaluating each of the variables and selecting the best or use the popular heuristic of choosing the variable with the maximum spread.
The node structure can be represented as

```
struct kdtree{
     int split_index;   /* for identifying the split variable*/
     float splitval;
     centre corner1;    /* specifying the minimum corner of the hyper rect
     centre corner2;    /* specifying the maximum corner of the hyper rect
     int n_eff;          /* records the number of effective RBF units */
     char *list;        /* maintains the list of effective units */
     struct kdtree *left;
     struct kdtree *right;
     }node;
```

The building of the radial basis tree can be done by calling the "build tree" function recursively as

```
Build_tree ( start_node ){
   if ( start_node.n_eff < threshold)exit;
   select split variable;
   n_eff_median = evaluate (median split);
   n_eff_wtdmean = evaluate (weighted mean split);
   if ( min( n_eff_median,n_eff_wtdmean)==start_node.n_eff)exit
   if( n_eff_median<n_eff_wtdmean)splitval= median split;
   else splitval= weighted mean split;
   allocate (start_node.right);
   allocate (start_node.left);
   assign node.right.corner1,assign node.corner2;
   assign node.left.corner1, assign node.corner2;
   build_tree( start_node.right );
   Build_tree( start_node.left );
   }
```

Chapter 5

Learning to shoot to goal

Shooting to goal, which is an important behaviour for a soccer playing agents, was chosen as the first task to be learned, for this study. Radial basis functions were used as function approximators. We have consider only the shooting angle to be the parameter to be learnt, assuming the shooter kicks with the highest velocity. The task is simple in so much so as only a single output has to be learnt and that this being a single epoch event does not involve the problem of temporal credit apportionment. However the input space dimensionality is still quite large, of the order of 10^{32} and poses difficulties for this problem.

For a shot aimed within an un-tended goal there is a finite probability that the goal will be missed, this is compounded with the additional problem of intelligent and reactive defender agents trying to stop the goal from being scored.

Goal scoring can be defined as the intersection of two mutually independent events i.e. that of ball remaining within the goal and that the ball is not intercepted by any of the defenders including the goal-keeper.

$$P(goal) = P(ball\, within\, goal) * \overline{P}(ball\, intercepted)$$

The probability of the ball remaining within goal can be estimated as

$$P(ball\, within\, goal) = \int_{left\, post}^{right\, post} p(\theta) d\theta \tag{5.1}$$

see Figure (5.1).
The probability of the ball not being intercepted can be expressed as

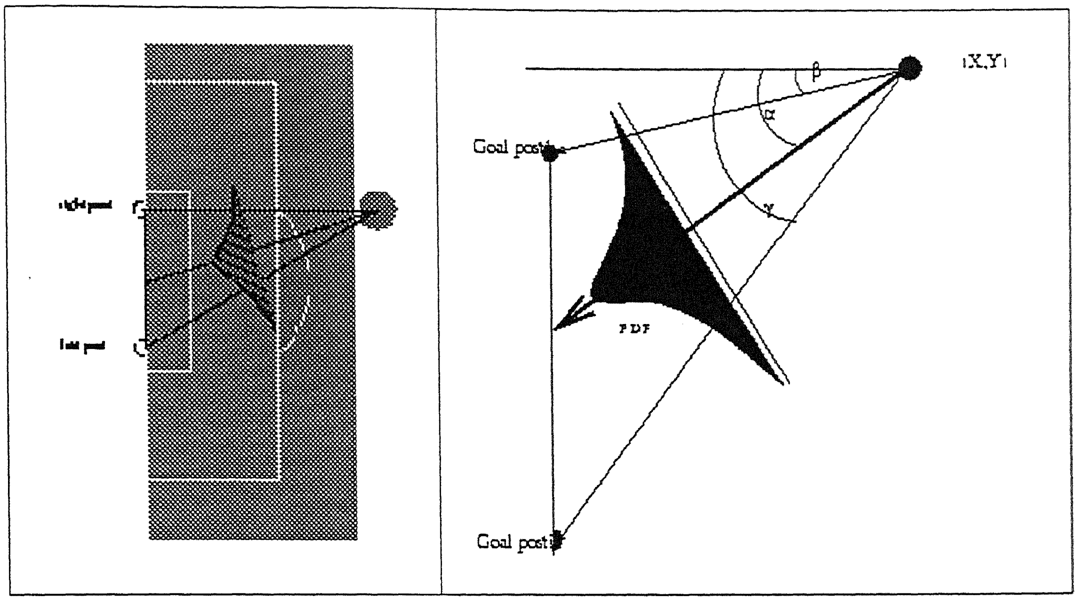


Figure 5.1: Noise in shooting angle

$$\overline{P}(ball\,intercepted) = \prod_{defender_i} (1 - P_{int(i)}) \quad (5.2)$$

Where $P_{int(i)}$ is the probability that the i^{th}defender intercepts the ball.

The task, thus is to maximize the probability of scoring goal in presence of adversaries and the random perturbations introduced by the soccer-server, by choosing a suitable shooting angle θ.

The task of learning the optimal shooting angle was learnt through supervised learning using multiple simulations to obtain the optimum value. Together with the optimum angle the maximum probability of a successful goal shot was also learnt. The latter is nothing but the value function $\mathcal{V}(\mathcal{S})$ of the state and can be updated online as discussed in section (2.2.1) and (3.3). The RBF network was tuned to approximate these functions.

5.1 Generation of training through simulation

For this particular case the state consists of the shooter (assumed to be in possession of the ball) , the goal keeper , and the two possible defenders. The

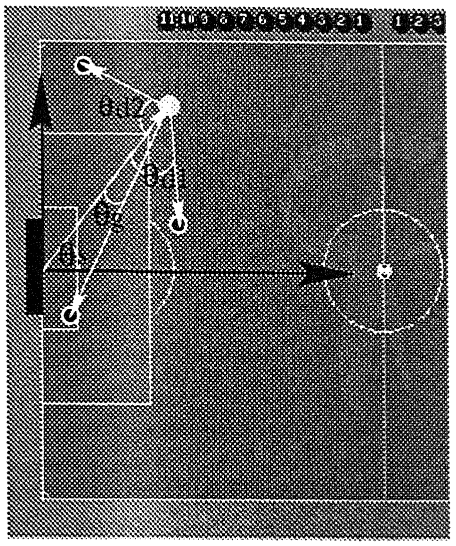


Figure 5.2: state variables of the shooting task

state $\mathcal{S}$ is thus represented as an eight tuple

$$< d_s, \theta_s, d_g, \theta_g, d_{d1}, \theta_{d1}, d_{d2}, \theta_{d2} >$$

where d is the reciprocal of the distance from the shooter (for the goal keeper and the defenders) or from the origin (for the case of the shooter). θ corresponds to the angle subtended at the foot of the shooter by the rays connecting the opponent and the origin, for the case of θ_s it corresponds to the angle subtended by the shooter at the origin see figure (5.2). and the action $\mathcal{A}$ is represented as a real number β in the range $[0, 1]$ representing the shots to the left goal post to the shots to the right goal post.

With a state consisting of eight real numbers the state space is huge. The enormity of the search space precludes exhaustive enumeration with a practical resolution. Thus it is necessary to distribute the training resource judiciously over the search space. Random initialisation of the training instance though tempting has been found to offer poor results [16].

5.2 Forming the training set:

The state space being large it is not possible to populate it evenly with training instances, with sufficient density. Training instances have been allocated

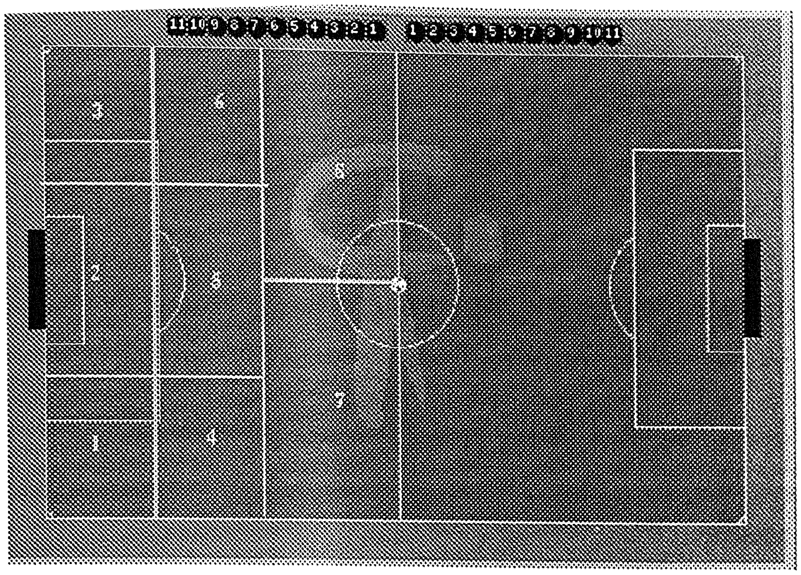


Figure 5.3: Seeding Training Instances

to the search space depending on the likelihood of the shooter experiencing such cases and their relevance to the output (shooting angle). For instance it is intuitive that a defender behind the shooter can very rarely intercept a shot so such redundant training cases have been omitted as far as possible.

As shown in the figure (5.3) the half field as been broken into eight celled grid. Training cases were generated by first placing the shooter randomly within all the segments. Following which the case of a defender was considered. Here the shooter is placed randomly in any of the columns and the position of the defender is so generated so that the defender is not placed on a column behind the shooters column. Similarly for the 2nd defender. In all the cases the goalie if present is located randomly in a small area within the penalty box . Ten shooting instances were generated for each configuration resulting in around 600 training cases.

5.3 Evaluation of goal scoring probabilities

Given a shooting angle parameter β [0-1] and the adversaries on the field as described by the state S the probability of scoring a goal

$$P(goal/\mathcal{S},\beta) = P(within\,goal/\mathcal{S},\beta) * \overline{P}(intercepted/\mathcal{S},\beta)$$

The shot to the goal is represented by a shooting angle parameter β. A value of $\beta = 0$ refers to a shot aimed at the left goal post i.e $\theta = \theta_{leftpost}$ and $\beta = 1$ refers to a shot aimed at the right post i.e $\theta = \theta_{rightpost}$

The probability of ball remaining within the goal $P(within\,goal/\mathcal{S},\beta)$ and that the ball is intercepted are calculated separately through simulation. Each of these parameters have been implemented as functions.

5.3.1 Probability of ball within goal

Here the simulation is started with the ball being hit with the maximum velocity with the shooting angle under investigation. At every time step the state is updated using the model of the soccer-server till the ball reaches the the goal line or decays to a halt. At the end of this epoch depending upon whether the ball was within the goal post or not the probability is updated as

$$P(within\,goal/\mathcal{S},\theta)_{t+1} + = (1 - P(within\,goal/\mathcal{S},\theta)_t) * 1/t$$

or

$$P(within\,goal/\mathcal{S},\theta)_{t+1} - = P(within\,goal/\mathcal{S},\theta)_t) * 1/t$$

where t is the number of iteration. This is continued till convergence. The variation of this value for different angles of shot, from a fixed position are shown in the figure (5.4).

5.3.2 Probability of interception

This routine has to take into account all possible behaviours of the defenders making it computationally more intensive than the previous function. As before, the simulation is initialised with the shooter shooting the ball with a specified shooting angle. We need to find out the probability that the ball can be intercepted by any of the defenders at any point on the trajectory of the shot.

Normally we would have carried out numerous iterations to find this probability. But since the number of dependent variables is much higher, the

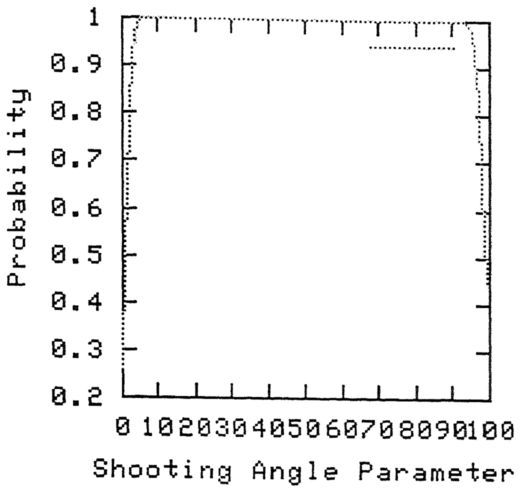


Figure 5.4: Scoring probability for an untended goal, shooting from (10,2).

number of iteration necessary to find the probability, taking into account all the possible variations of the variables ,is expected to be huge. So a different strategy is adopted to reduce the computation.

We define $p_{ball}(d, t)$ as the probability distribution of finding the the ball at a distance d from the point of kick in time t .

Thus the probability that the ball is at a distance d from the point of kick in the t^{th} time interval is obtained as

$$\int_t^{t+1} p_{ball}(d, t)dt$$

For the discrete model of the soccer server this information is approximated as a piecewise constant curve and stored as a sparse matrix $p_{ball}[d][t]$ for an adequate range of values of t and d .This pre-calculation saves on generating these values at each iteration at the expense of memory.

Similarly for the defenders

$p_{def_i}(d, t)$ is the probability distribution of finding the i^{th} defender at a distance d from his initial point in time t . It is assumed that the defender is running with the maximum possible speed.

This information is stored subject to the piecewise constant assumption as a sparse matrix $p_{def_i}[d][t]$.

The entire matrix can be visualised as a 3 dimensional surface built by stacking these curves along the distance axis , as shown in Fig5.5 and Fig5.6.

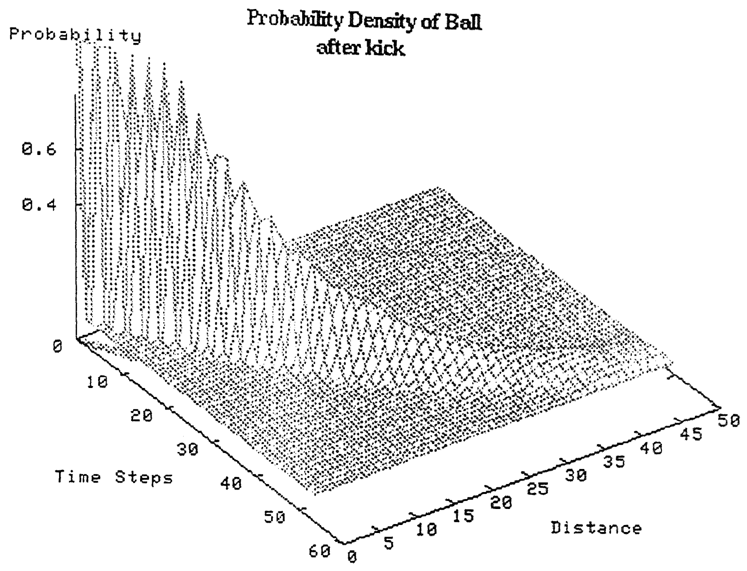


Figure 5.5: Probability of finding ball at a distance d at time t

The nature of the second plot is significantly different because the defenders can replenish the decay of their speed, and the noise added in their motion are lower than that of the ball.

For a given shooting angle θ the ball is checked for interceptability along all the points of the trajectory of shot. The points on the trajectory are given by the vector

$$\begin{bmatrix} x_l \\ y_l \end{bmatrix} = \begin{bmatrix} x + l \times \cos\theta \\ y + l \times \sin\theta \end{bmatrix}$$

where $\begin{bmatrix} x \\ y \end{bmatrix}$ is the initial location of the ball (identical to the location of the shooter) and l is a parameter.

For a given shooting angle θ , at every value of l we evaluate the distance $dist_i$ of all the defenders from the point $\begin{bmatrix} x_l \\ y_l \end{bmatrix}$. This distance is used to find the probability that the defenders can reach it in time T. This is calculated as

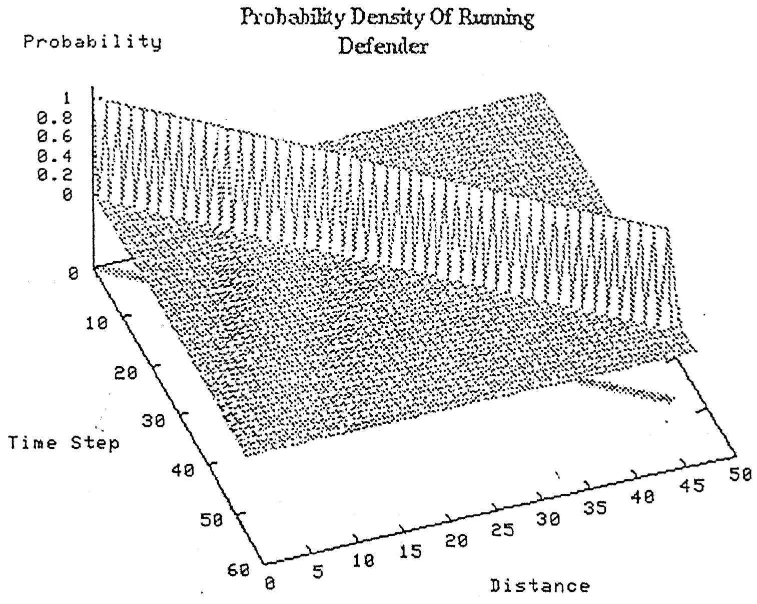


Figure 5.6: Probability of finding defender at distance d at time t

$$P_{def_i}(dist_i, T) = \sum_{o}^{T} p_{def_i}(dist_i, t)$$

similarly for the ball we calculate

$$P_{ball}(l, T) = \sum_{0}^{T} p_{ball}(l, t)$$

Thus the probability of intercepting the ball at $[\begin{smallmatrix} x_l \\ y_l \end{smallmatrix}]$ in time T is given as

$$P_{int}(l, T, \theta) = P_{def_i}(dist_i, T) * \overline{P}_{ball}(l, T)$$

To find the probability of interception of the ball at $[\begin{smallmatrix} x_l \\ y_l \end{smallmatrix}]$ we have to consider the probability for T ranging from 0 to ∞. We take this probability

$$P_{int}(l, \theta) = \max_{T} P_{int}(l, T) \tag{5.3}$$

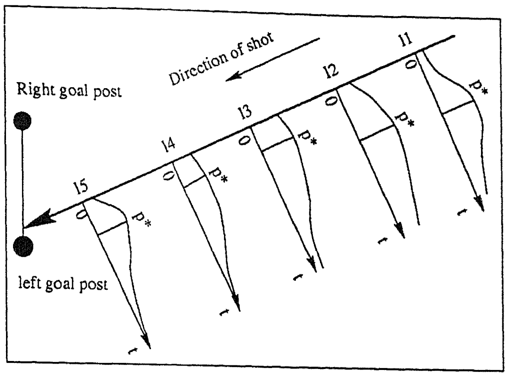


Figure 5.7: Schematic diagram showing maximum interception probability over time at different positions along the direction of shot.

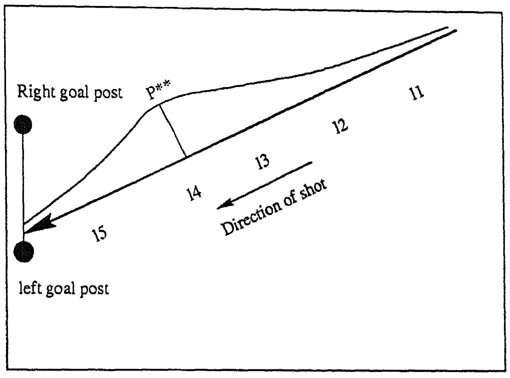


Figure 5.8: Schematic diagram showing maximum interception probability over the length of shot. Each point corresponds to the maximum obtained from the earlier figure (5.7).

This quantity is evaluated at all the values of l till the goal is reached and the probability that the ball is intercepted for the given shooting angle θ is calculated as

$$P_{int}(\theta) = \max_{l} P_{int}(l) \tag{5.4}$$

The probability of interception taking into consideration all the defenders is given by the expression (5.2). The max operators in the equations (5.3)

no:of centres	clustering	mean absolute error in angle	cross-validation σ_{GCV}	$\frac{rmse}{mad}$
60	hard k-means	.142881	.23050	1.461
120	hard k-means	.085057	.208490	1.950
60	soft k-means	0.175498	0.272719	1.381
120	soft k-means	0.132576	0.275045	1.616
60	gradient descent	0.102704	0.205560	1.778

Table 5.1: Result of training of optimum angle

and (5.4) achieve the ideal opponent modelling.

5.4 Implementation and results

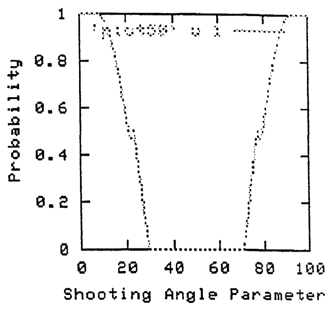
The plots of the probabilities of goal scoring generated by simulation as explained in section (5.3) are shown for different cases in the figures (5.9, 5.4). They show the effect on widening the "goal window" as the shooter comes nearer to the goal. The effect of the side, which the shooter and the goal keeper is on is also shown on the next to plots (5.10).

The RBF network was trained to learn the optimum shooting angle considering at most three defenders including the goal keeper. The input was the 8 tuple indicated in section [5.1] the output was a parameter ranging from 0 to 1 signifying a shot to the left goal post to a shot directed at the right goal post.

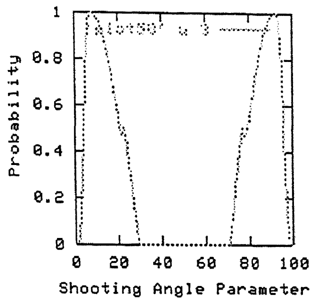
It was decided that an angular accuracy that could divide the goal line into 5 blocks were sufficient for the purpose. Thus an absolute error in angle value less than .2 was enough, but a larger network with 120 centres was also investigated to see its effect on the training instances. The table [5.1] gives the values of the absolute and generalized cross validation error for the networks trained with different number of centres. Note the number of centres signify the effective number of states the problem space was broken into.

In addition to these angle values the value functions, which in this case was same as scoring probability was also learnt. The results of which are shown in the table [5.2].

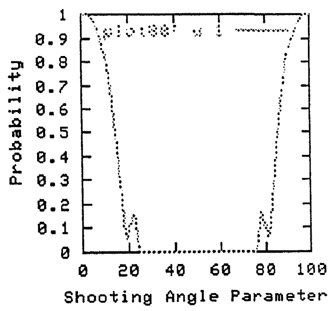
The hard k-means clustering is scene to be performing better for this



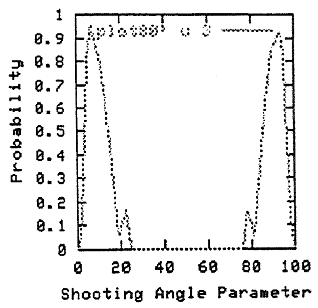
(a) Probability of beating the goal-keeper stationed at the centre of the goal line by a shooter 5m in front.



(b) Goal scoring probability for same configuration as on the left

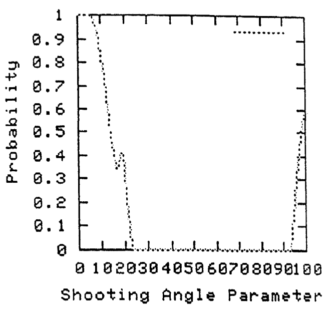


(c) Probability of beating the goal keeper stationed at the centre of the goal line , by a shooter at 8m in front.

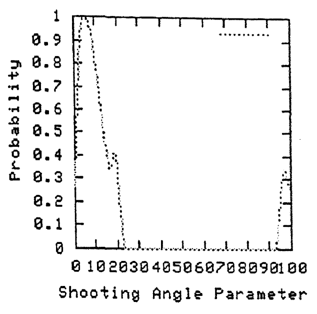


(d) Goal scoring probability for the same configuration as on the left

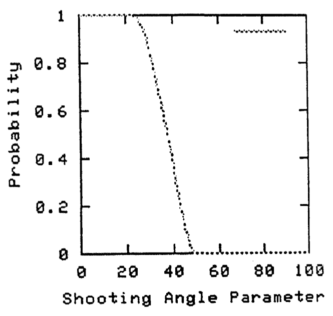
Figure 5.9: Scoring probabilities from various locations I



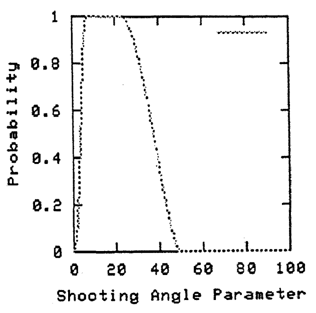
(a) Probability of beating the goal keeper stationed at the centre of the goal line by a shooter 10m in front and 2m to the left



(b) Goal scaoring probability from the same configuaration



(c) Probability of beating the goal keeper for the same location of the shooter as above, but with the goal-keeper 2m on the righ



(d) Goal scaoring probability for the same configuaraion

Figure 5.10: Scoring probabilities from various locations

no:of centres	clustering	mean absolute error in probability	cross-validation σ_{GCV}	$\frac{rmse}{mad}$
60	hard k-means	.143372	.342407	2.122
120	hard k-means	.072632	.249786	2.627
60	soft k-means	0.155177	0.355177	2.03
120	soft k-means	0.121895	0.346897	2.217
60	gradient descent	0.068272	0.230501	3.00

Table 5.2: Result of training scoring probabilities

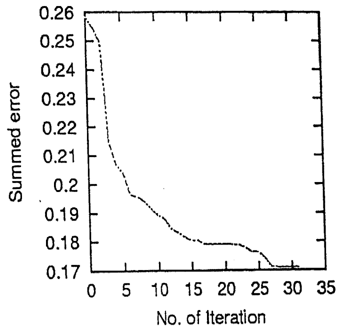


Figure 5.11: Error reduction by steepest gradient after hard k means clustering.

particular domain. Poor performance of soft clustering can be attributed to the higher degree of generalization or "smearing effect" produced by soft clustering. The centres obtained by hard clustering were further enhanced by a gradient descent algorithm, for the case of 60 centres. The plots of which are shown in figure (5.11). For the case of 120 centres the error levels were regarded to be sufficiently low to make gradient descent unnecessary. It was observed that the gradient descent did not change the location of the centres $\bar{c}$ by much, in other words the centre locations produced by the k-means clustering are quite close to the optimum. The better performance of the gradient descent is the outcome of tuning the width parameters σ , which initially had been set proportional to the radius of the clusters.

One of the advantages of RBF networks is that they produces a comprehensible model. The rules (or centres) learnt for the optimum angle case are shown pictographically in the figures(5.12 to 5.17). The shooter is represented by a light coloured circle, the opponents are shown as a white ring around a black circle. The optimum shooting angle as proposed by the rule is shown by a directed straight line. The figures have been arranged roughly with an increasing level of opposition. One of the things that can be noted is the rules are more extremal, for example in the figure (5.12 bottom) the rule proposes a shot to the goal post. This can be explained by the fact that these rules are trying to compensate for the fact that the final output being averaged over all the other rules will get somewhat moderated. Moreover the effect of the goal posts are localised to a fairly small area around each goal post figure (5.4) making near goal post shots very effective.

The RBF tree was implemented both for the case with 120 and 60 centres. The generated tree structure (all the nodes have not been fully expanded for brevity) is shown in figure(5.18).Partitioning was stopped for nodes with n_{eff} equal to 5.

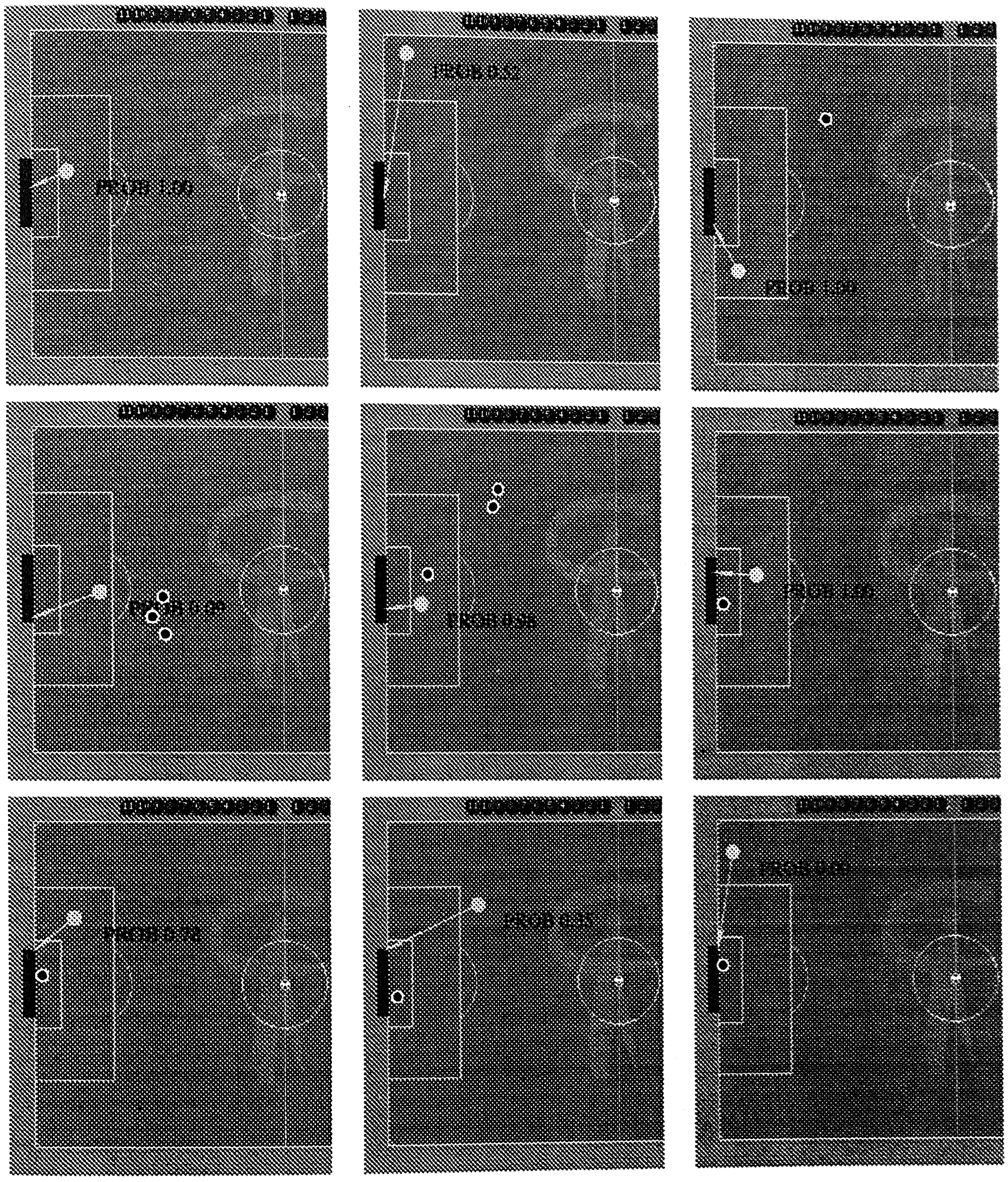


Figure 5.12: Learnt Rules I

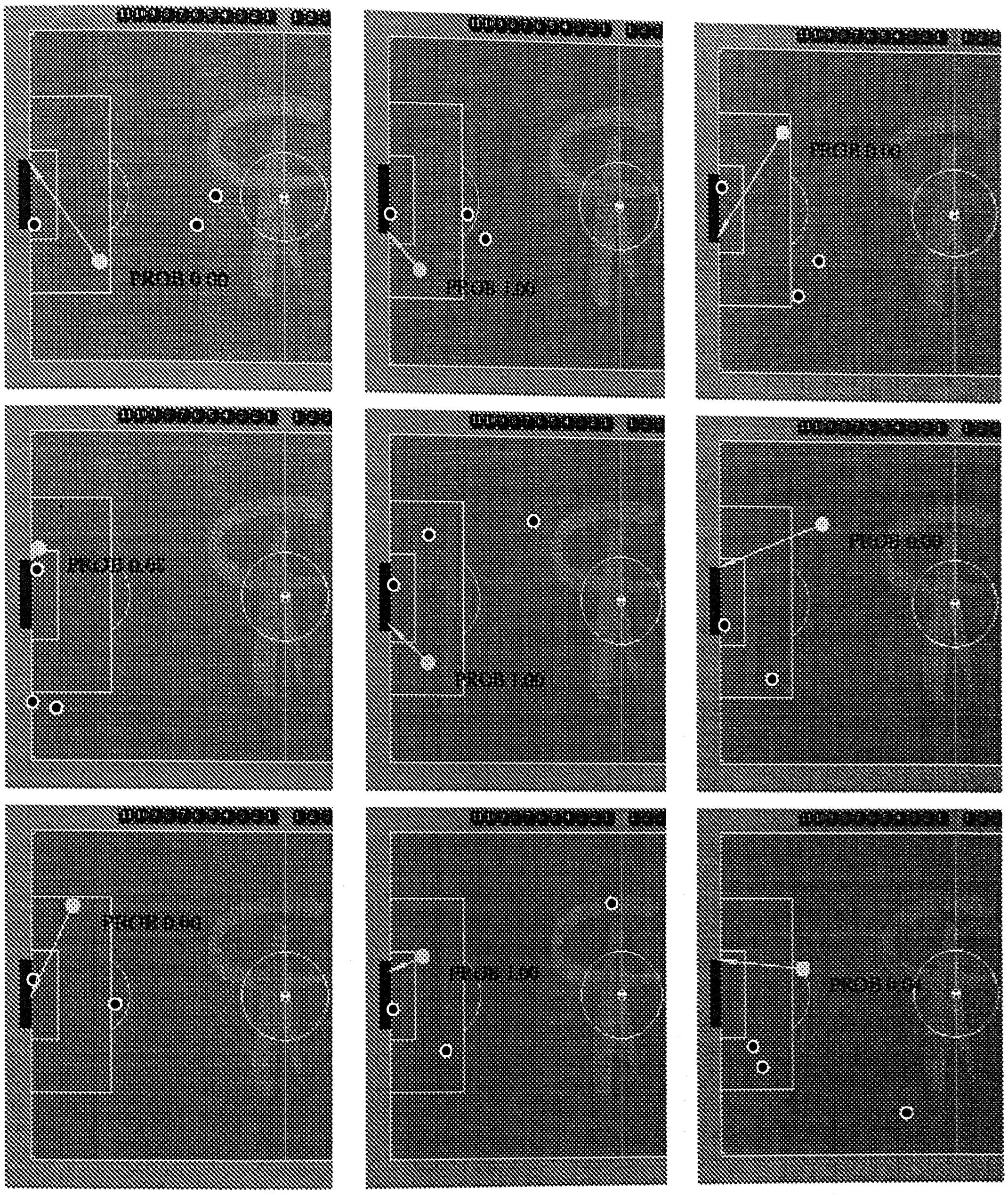


Figure 5.13: Learnt Rules II

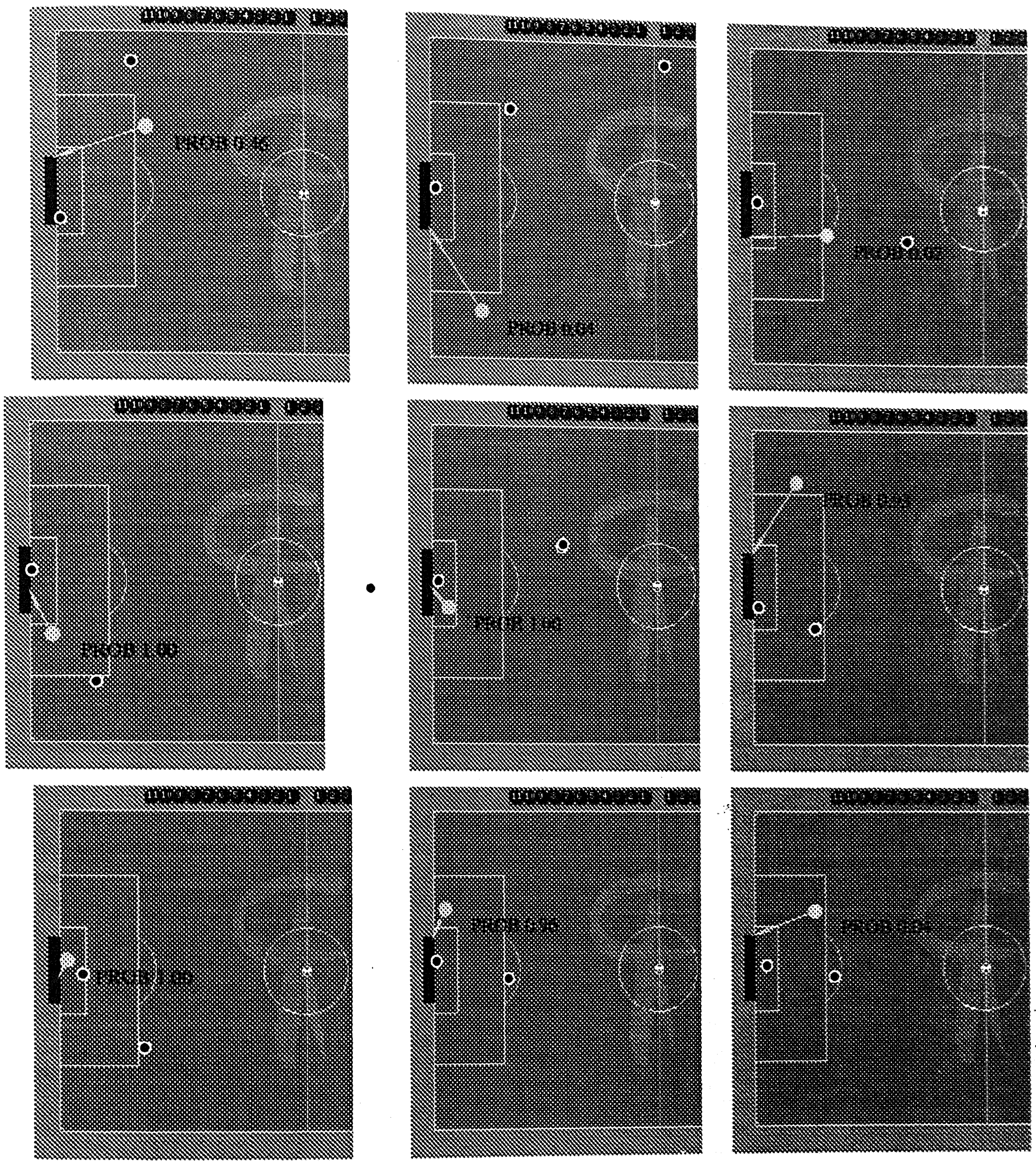


Figure 5.14: Learnt Rule III

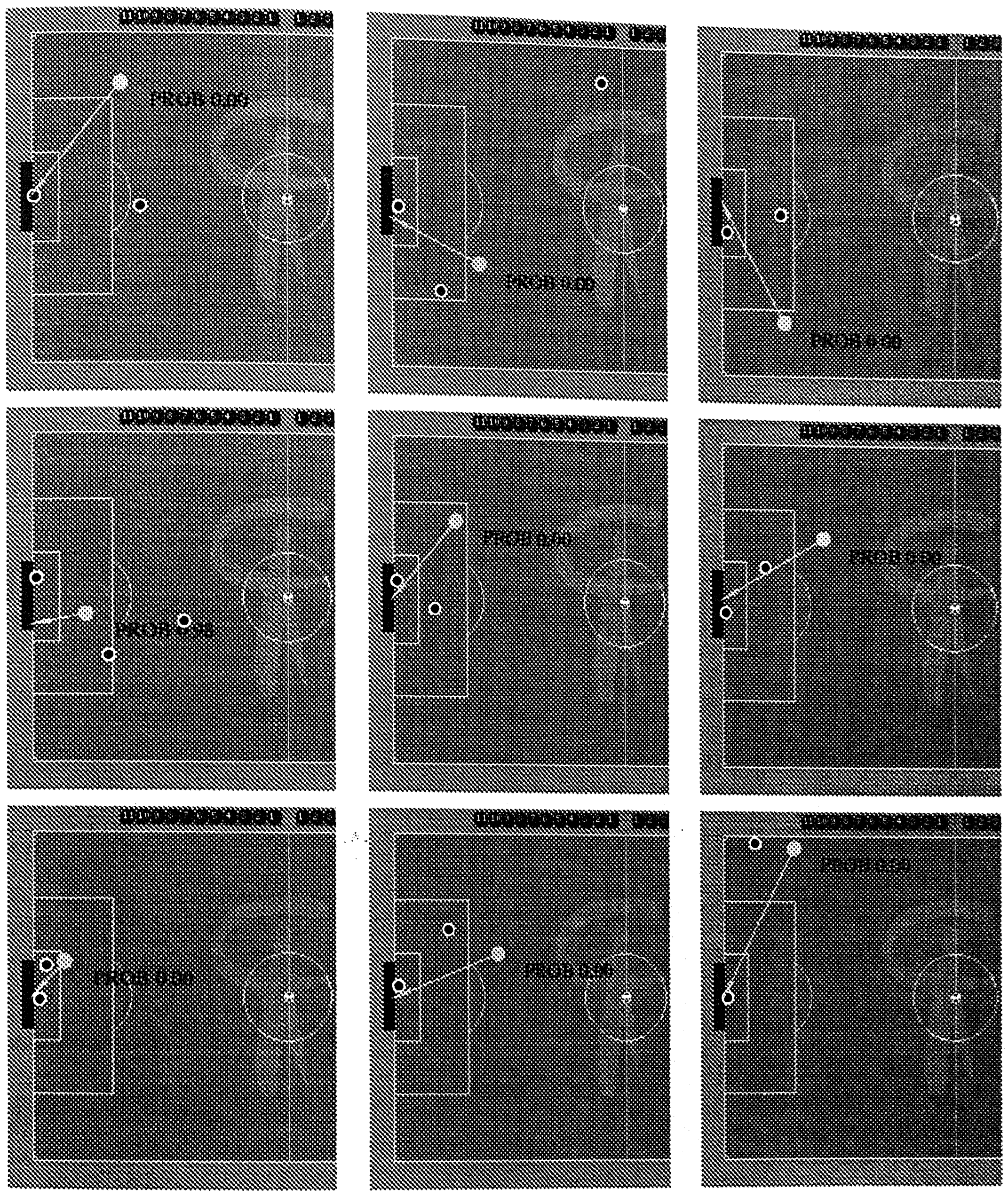


Figure 5.15: Learnt Rules IV

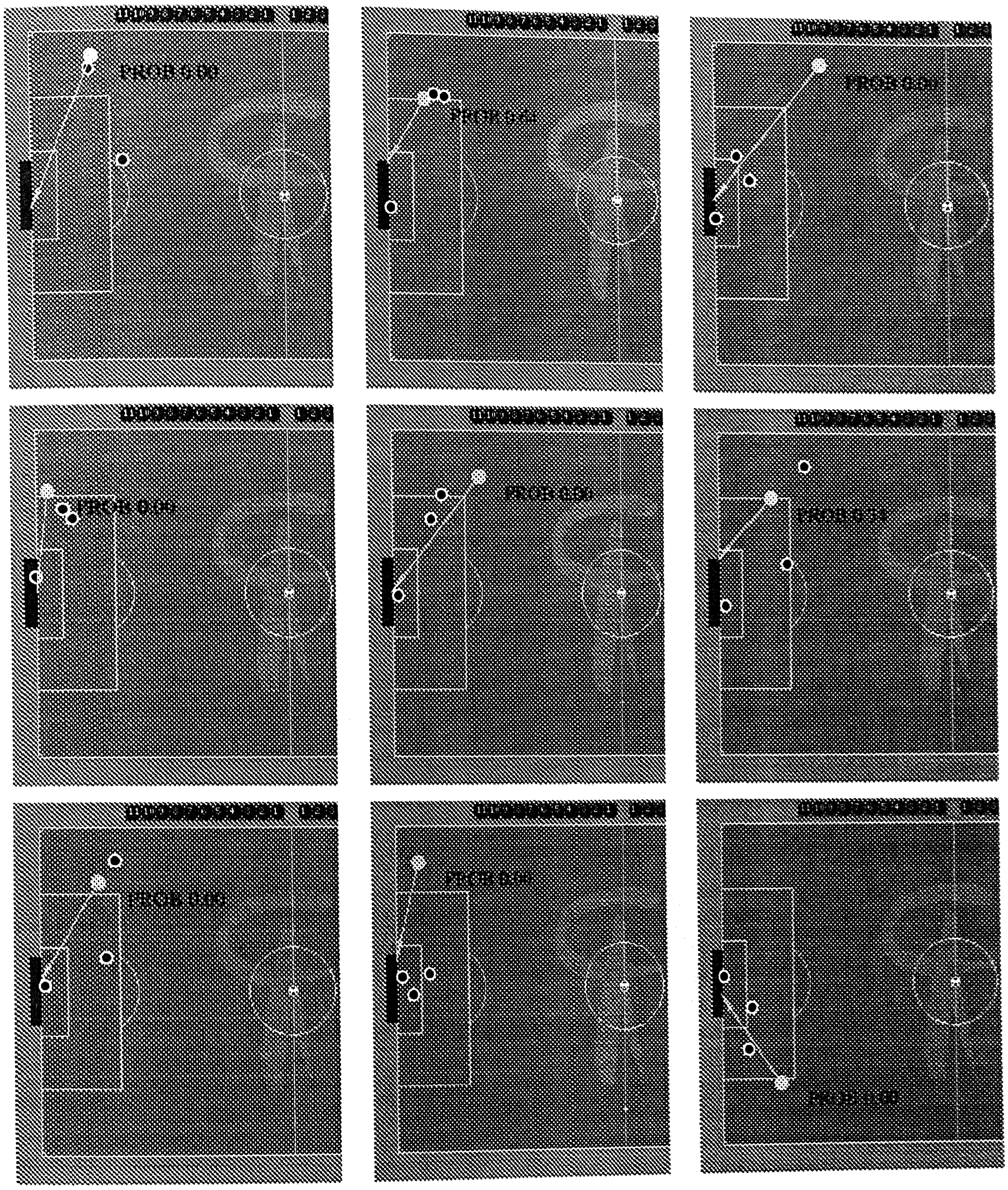


Figure 5.16: Learnt Rules V

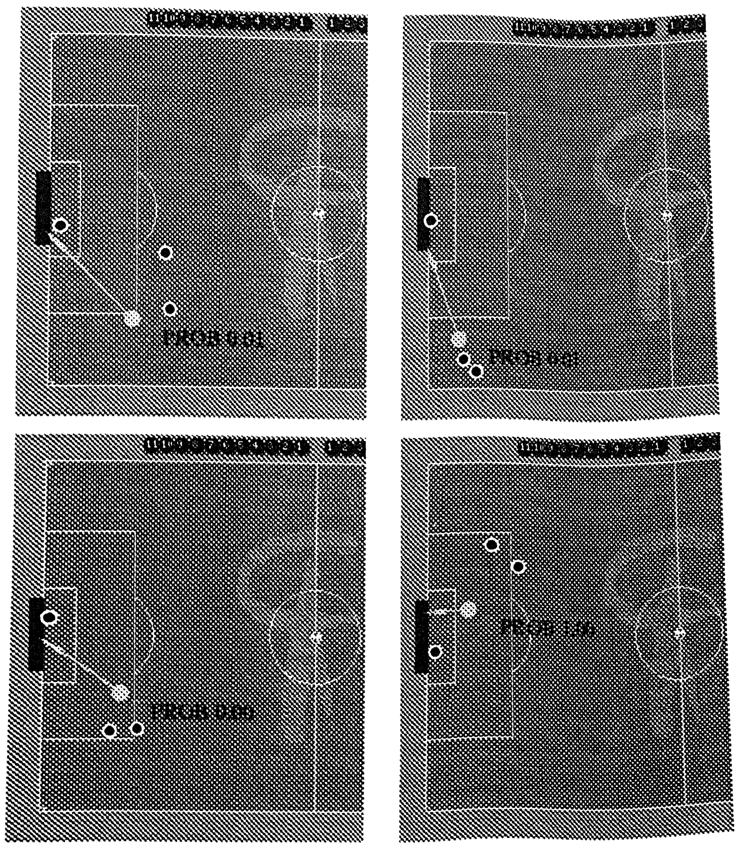


Figure 5.17: Learnt Rules VI

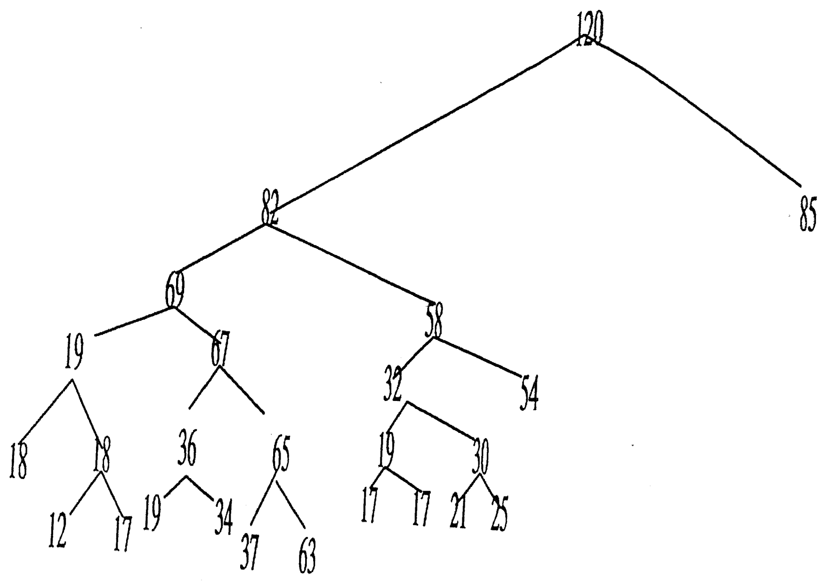


Figure 5.18: A Partial view of the RBF tree generated for the case of 120 centres. The figures represent the number of effective units at the nodes.

Chapter 6

Learning to intercept ball

Interception of moving ball constitutes an important skill in the robot-soccer domain as it is necessary for a number of collaborative as well as adversarial tasks like receiving a pass from a team member, saving a goal, stealing ball from opponent etc. This task can also be directly related to that of choosing optimal passing shots in which case the desired speed and direction of the pass is decided based on the interception probability by ones teammates. However this task has not been adequately studied in the past. Interception has been learnt mainly for cases where the ball is directed straight to the interceptor [6] and the presence of optimal opponents have been neglected.

6.1 Input vector and the generation of training cases

The input vector considered for this task consisted of a 5 tuple $< d_b, v_b, \theta_b, d_o, \theta_o >$ where the variable d correspond to the reciprocal of the distance from the agent (or origin for d_b), the variable θ corresponds to the angular distance see figure (6.1). The subscript b refers to the ball and subscript o refers to the opponent.

In learning the interception task the variables d_o and θ_o where taken in the reference frame of the agent, where as for the passing module it can be taken in the frame of the ball i.e. the passer.

A total of 6600 training cases were generated by considering different combinations of these variables. Each of the variables were distributed in a noise contaminated geometric progression between the highest and lowest

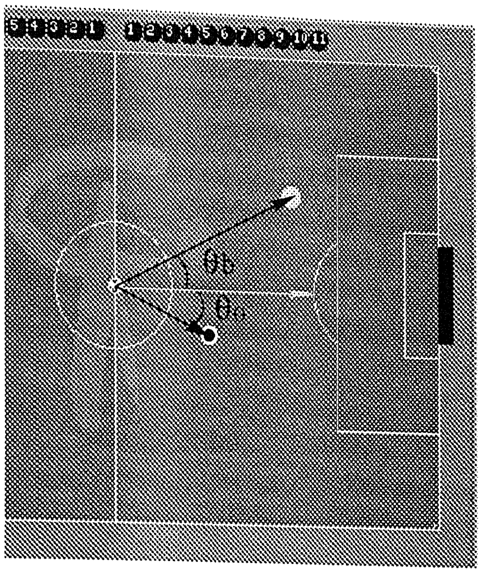


Figure 6.1: State variables of interception task

values fixed a priori.

6.2 Evaluation of interception probability and Q values.

Instead of simulating each of the 6600 cases thousands of times the same probabilities computed in [5.3] were utilized to arrive at the probabilities of interception and the associated Q values.

The action space A_i consisted of 10 *macro actions* numbered 1-10 for 3 different interception velocities of the agent (3m/s, 6m/s , 10m/s). The action ${}_vA_i$ i.e. the i^{th}action with velocity v referred to intercepting the ball at the location where it is expected to be at the i^{th} time step, i , as mentioned could take any value from 1 to 10. These actions are called macro actions as they continue to be executed till their exit criteria is reached, as opposed to ordinary actions which are selected at every time step. This macro-action representation has been found to significantly expedite the learning process [31].

6.2.1 Rewards and penalties:

The behaviour learnt by the agent is entirely guided by the incentives provided in the form of the reward function. The required behaviour for this task demanded that the agent should intercept the ball before the opponent. In this case the opponent was modelled to behave optimally. Thus a reward of 1 was given for a successful interception and a penalty of .9 was given in case the opponent captured the ball before the agent.

The magnitude of the penalty has been kept lower than the reward because if both of them were equal then there would be no incentive for the agent to try an intercept the ball which the opponent is more likely to capture. Thus by reducing the magnitude of the punishment the agent was given a weakly optimistic view of the world. It was also desired that the agent capture the ball in minimum time. The soccer server penalizes each dash by a value corresponding to the strength of the dash. Thus a penalty inversely proportional to the dash velocity v_d was applied.

The probability that the agent can intercept the ball was found for every point on the ball's trajectory till 10 simulation steps, similarly for the defender. Thus in terms of these probabilities $p_{intercept\,agent}$ and $p_{intercept\,opponent}$ the net Q value could be calculated as

$$\mathcal{Q}(\mathcal{S}_{,v_d}\mathcal{A}_t) = \gamma^t(p_{intercept\,agent} - 0.9p_{intercept\,opponent}) - \frac{(1-\gamma^t)(k/v_d)}{1-\gamma}$$

6.3 Implementation and Results

The radial basis network as trained to learn the q values of the macro actions. The results are summarized in the table [6.1].

Similar to the goal shooting case the pictorial representation of the behaviours learnt for intercepting the ball is included. Unlike the previous case here only the $\mathcal{Q}(\mathcal{S} \times \mathcal{A})$ model was learnt. The figures represent the the configuration for which the $\mathcal{Q}$ value is maximum.

As before the agent is in a light coloured circle whereas the opponent is represented by a light ring over a dark interior. The state space representation used for this task is based on the relative coordinates so is independent of the actual location of ball but for convenience the ball is assumed to be initiated from the centre. The light arrow signifies the distance the ball can cover in 10 time steps if it is not intercepted, and gives an idea of the veloc-

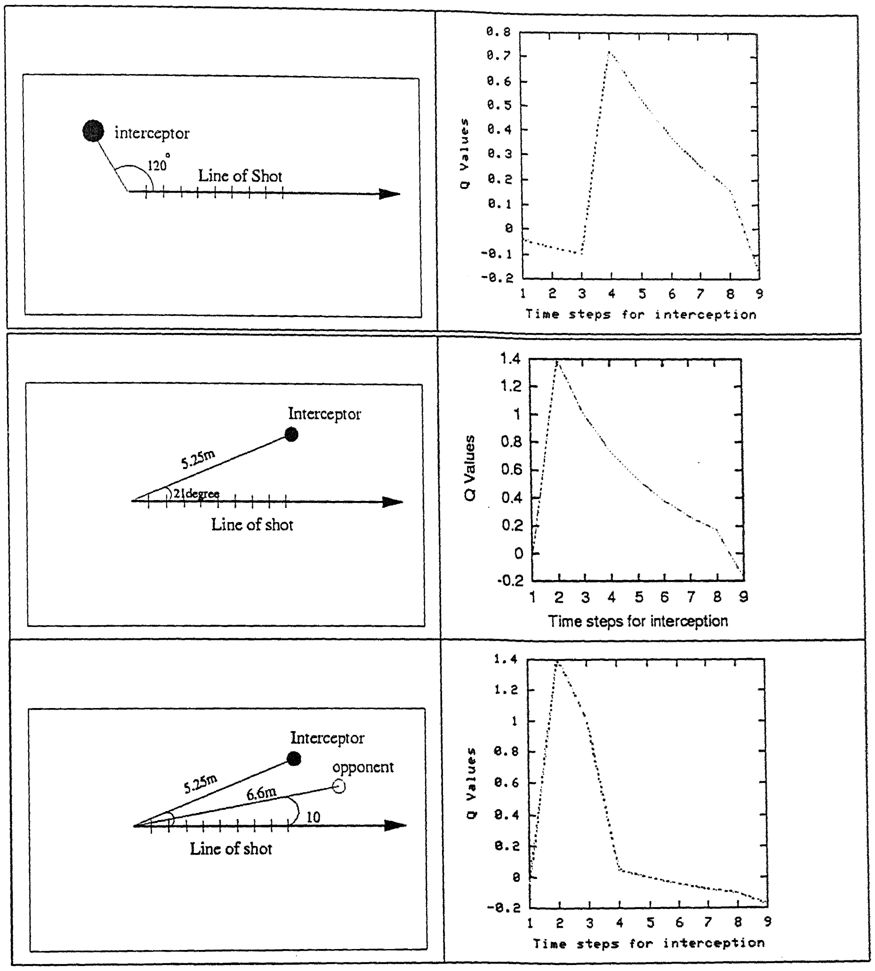


Figure 6.2: Plots of Q values for various configurations. For the bottom case note how the Q value falls of sharply because of the presence of an opponent.

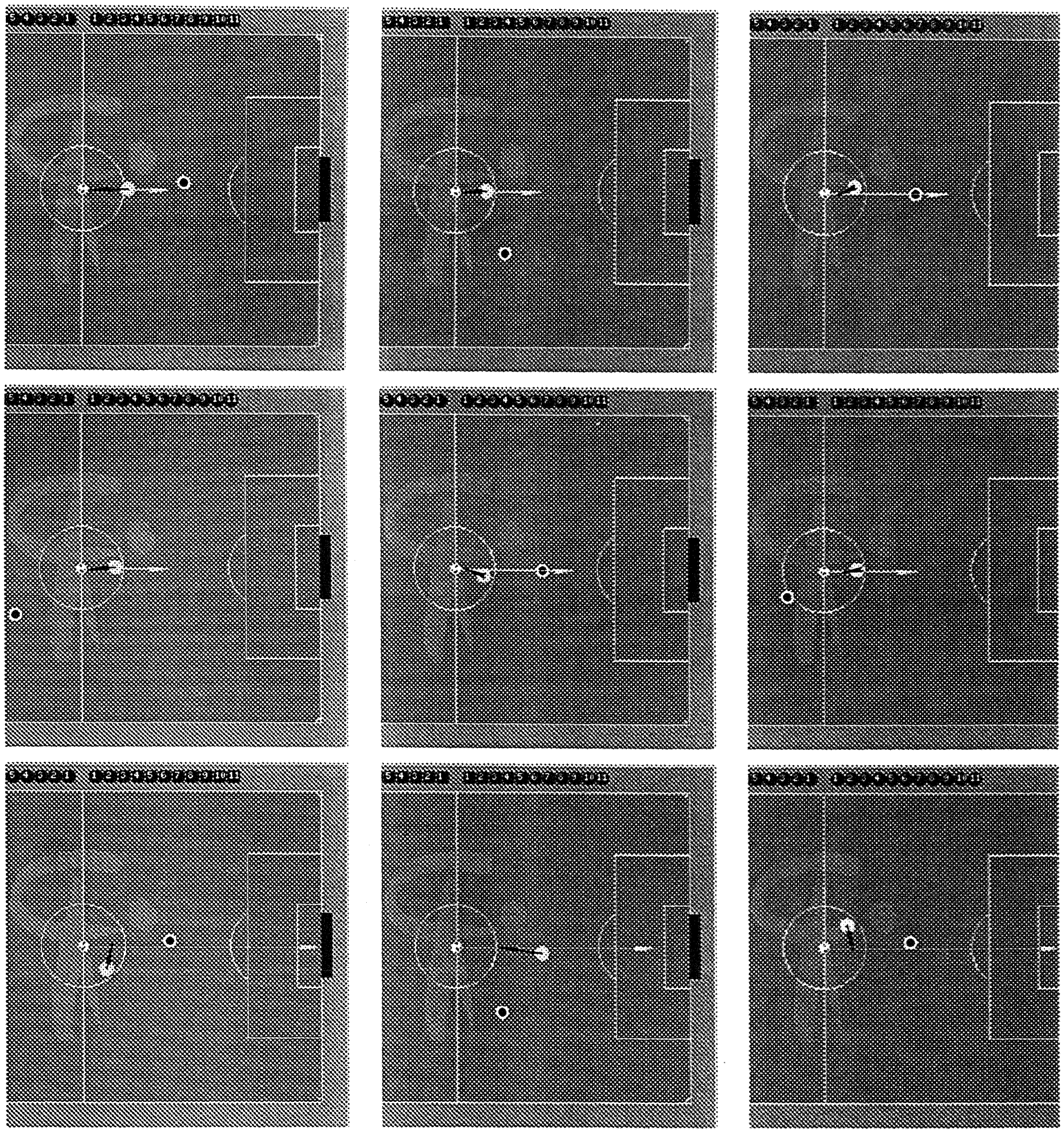


Figure 6.3: Rules Learnt for interception I

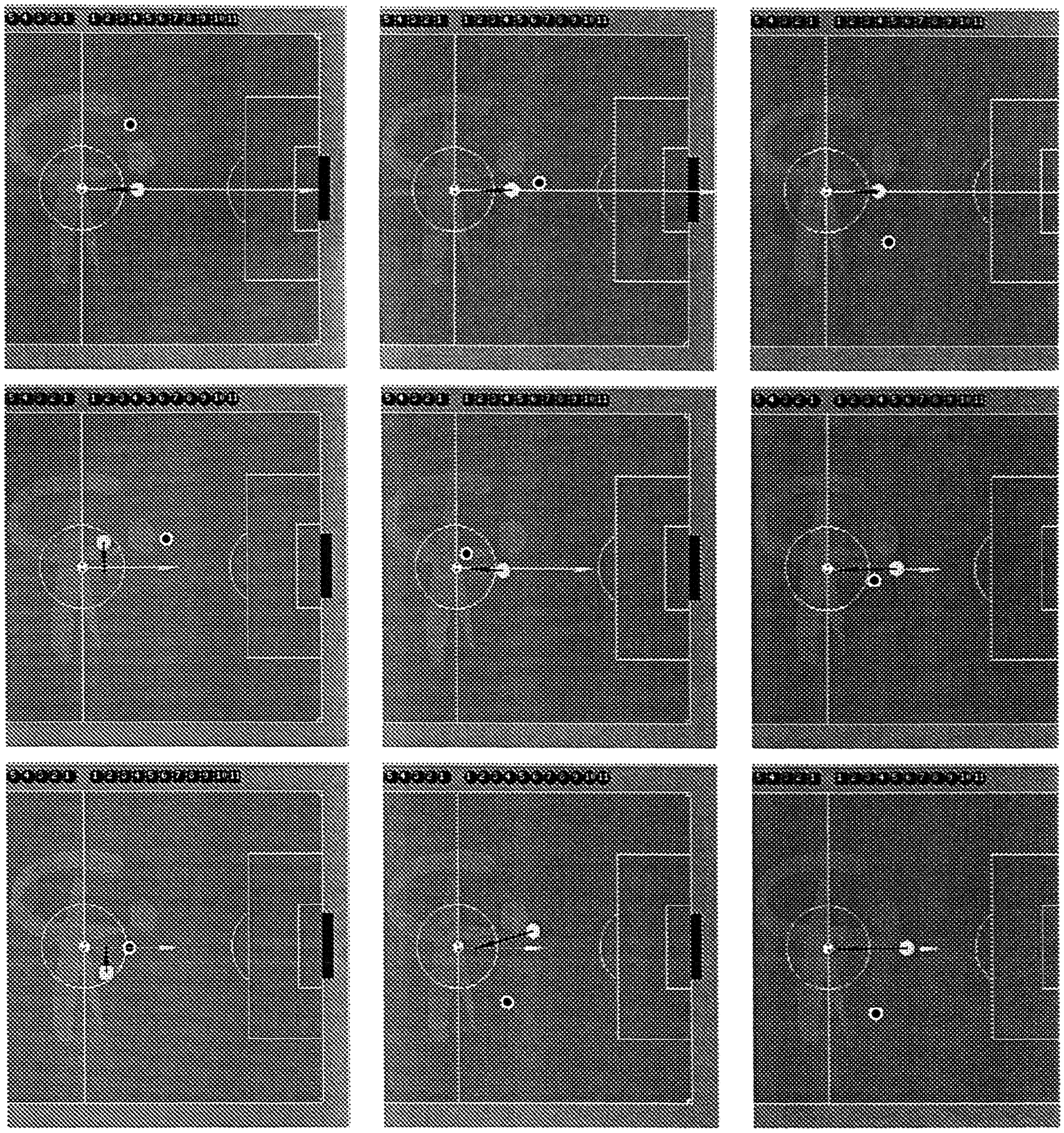


Figure 6.4: Rules Learnt for Interception II

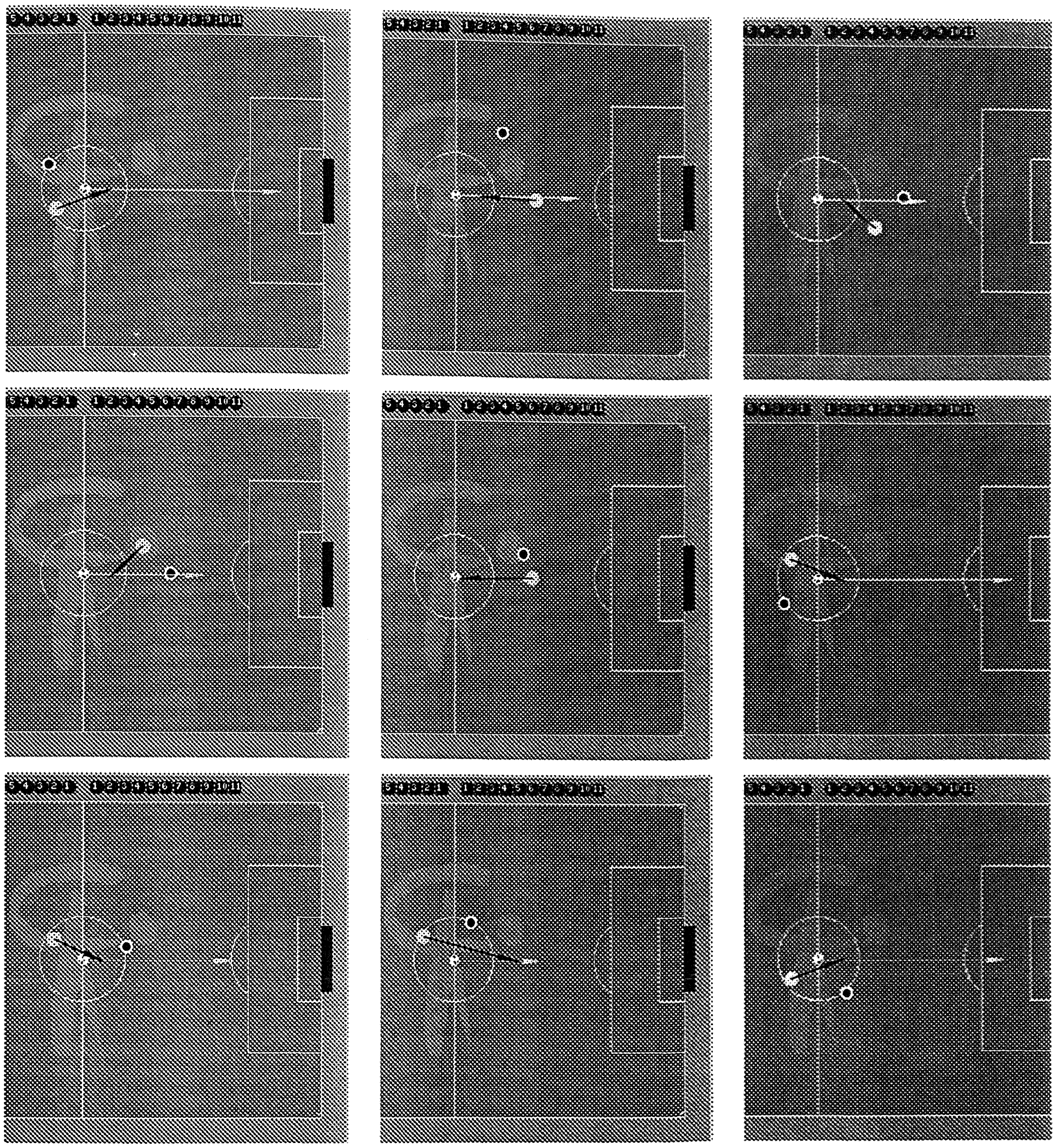


Figure 6.5: Rules Learnt for Interception III

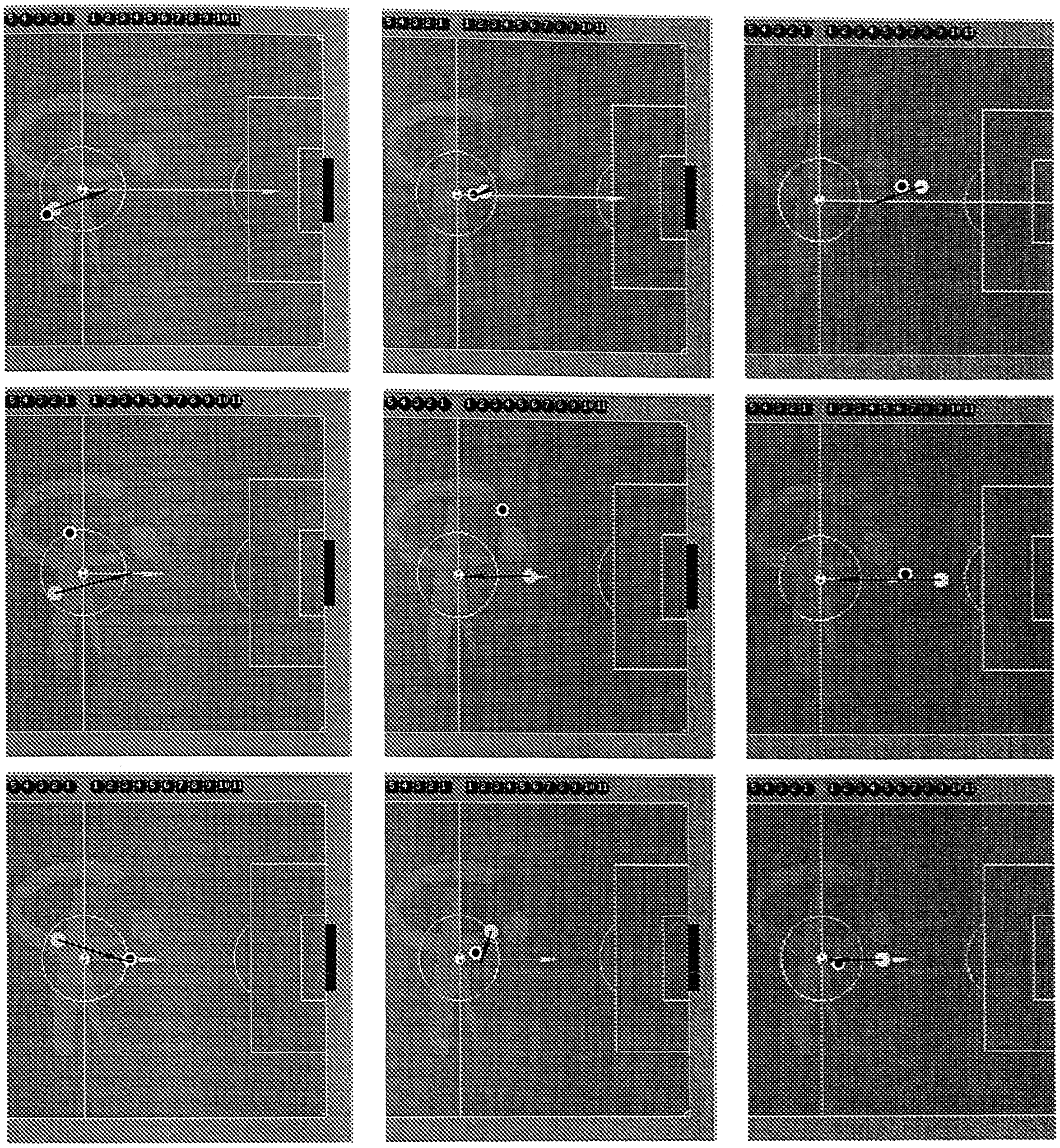


Figure 6.6: Rules Learnt for Interception IV

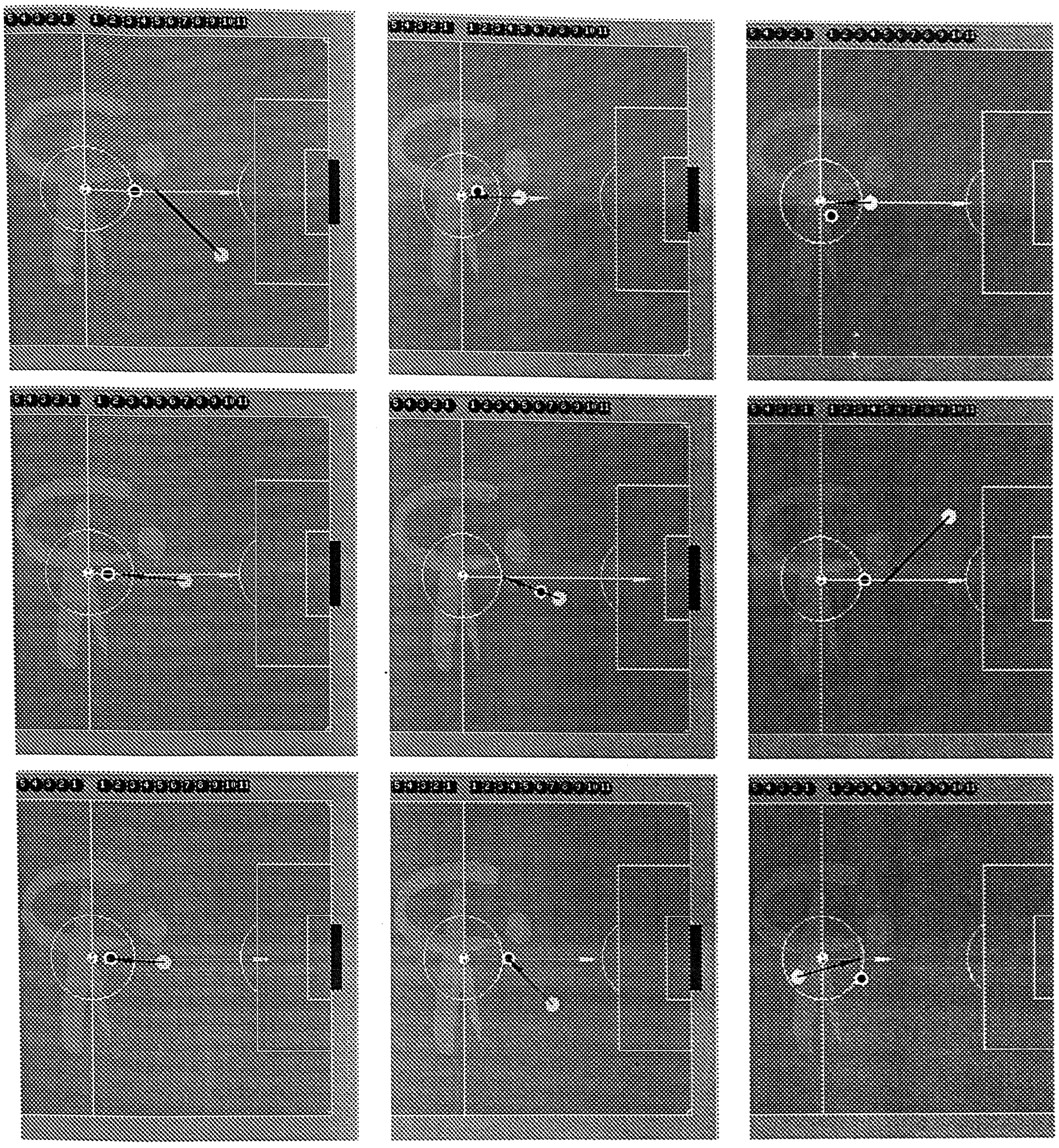


Figure 6.7: Rules Learnt for Interception V

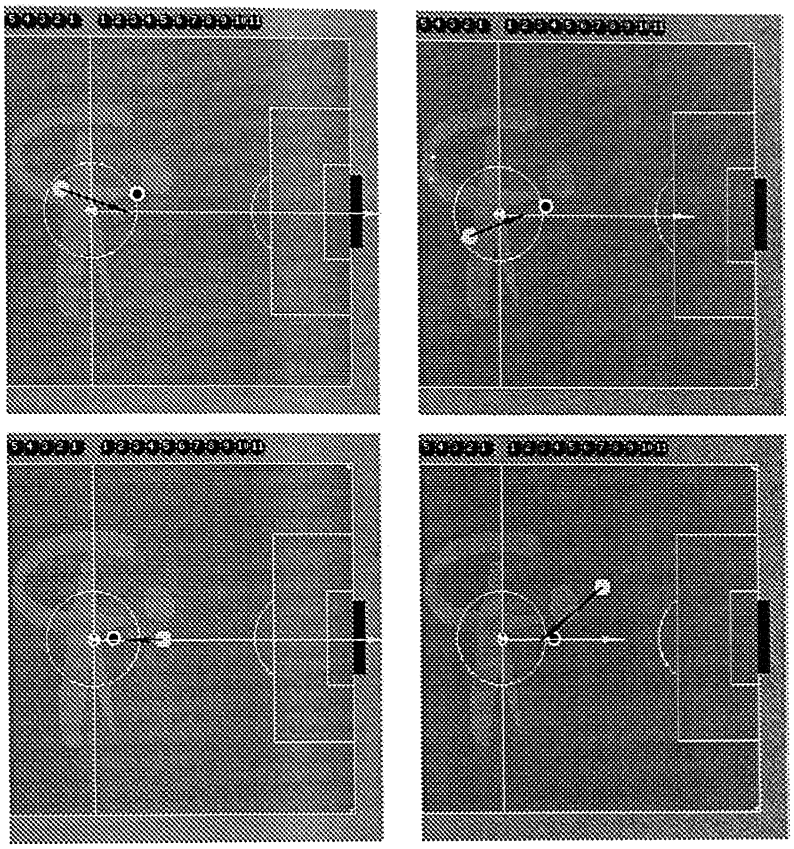


Figure 6.8: Rules Learnt for Interception VI

Chapter 7

Derived Behaviours

Following the subsumption architecture as explained in chapter (1) typical use of these modules as envisaged by a number of researchers [3] are to form the lower rungs of a complete learning agent see figure (7.1). A meta level gating function is required to activate the most appropriate behaviour depending on the state. The learnt modules can not only form these elemental behaviours but also serve as a basis for bootstrapping other novel behaviours.

7.1 Behaviours from goal Shooting task

The Learnt RBFN can act as a domain knowledge from which other important tasks can be derived. Among the skills that can be obtained from the goal shooting task, include optimal strategic position occupation by the goalie, and the defenders.

The learnt RBFN returns the optimal goal shooting point and the associated probability of scoring. This information can be used within the framework of an ideal opponent model (in this case the shooter) to select the best position for the goal keeper.

A function minimization method like gradient descent can be used to minimize the goal scoring probability. The method of gradient descent is specially attractive as RBFN model is analytically differentiable. The differential coefficients of the state vector $\bar{x}$ will be similar to that of the centre $\bar{c}$ already derived in section [3.5.4] see equation (3.13). A complete gradient descent procedure may prove to be expensive to undertake online, so one might resort to undertaking a fixed number of iterations of gradient descent.

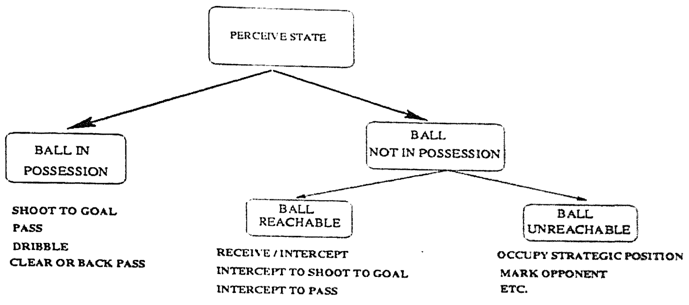


Figure 7.1: Hierarchy of Behaviours

As an approximation one might consider the RBF outputs to be constant in which case the minimality criteria decays to a linear equation in a single variable. The accuracy of this can be improved by repeated solution of the associated single variable linear equation. A much simpler solution can also be undertaken by calculating the goal scoring probabilities at a fixed number of goal keeper locations and selecting the minimum. The same strategy can be used to consider the position of the defenders near goal.

7.2 Behaviours from Interception task

An elemental skill derivable from the interception module include passing, This does not involve learning of any new concept, only the state has to be transformed to a passer centric model from the recipient centric model. In the current implementation a ball centric model was used for the RBF network. Depending upon the task desired any of the "recipient" or "passer" centric model has to be preprocessed to obtain a ball centric representation at the time of use.

A number of "stitched" tasks of the type "passing for X" and "intercepting for X" can also be undertaken with the learnt model. Here "X" can represent any other elemental subtask for example shooting to goal. In these cases the action selected should maximize the probability of success of the total task. For example while "intercepting to shoot to goal" the agent will try to intercept the ball where the total expected reinforcement i.e. from capturing the ball as well as shooting from that location is maximized. This quantity is nothing but the product of the output of the learnt models.

Apart from passing to another opponent, the agents may also choose to dribble forward with the ball. An agent dribbles by repeatedly kicking and intercepting the ball while moving towards opponent goal. It is desired that the agents penetrate opponent area as fast as possible without losing control of the ball. This requires the agent to kick hard when the threat of interception by opponents is low and vice versa.

This task also may be derived on the basis of the tasks already learnt, using the trained RBF networks to provide the necessary reinforcement for this multi-epoch task. For every action undertaken by the agent, the reward should consist of the goal shooting probability from the point reached by the ball, the probability of intercepting his own shot and a negative reward given on the basis of the interception probability of opponents.

Chapter 8

Conclusion

A novel approach of using RBF networks was used for the simulated soccer competition domain. The advantages offered being fast training time, comprehensibility in the form of rules, but most of all a powerful representation that can be modified incrementally (online) without distorting past training sets. This include not only addition of new data but even adding new centres or rules. In order to maintain optimality normally one needs to re tune the network from scratch, here the same can be undertaken by a computational effort lower by an order of magnitude. A tree based algorithm was also developed to increase the recall time, thereby making it feasible to utilise much larger RBF network models.

One of the issues that can be investigated in greater detail include the the choice of clustering method. The study implemented hard and soft forms of k-means clustering together with a gradient decent method for finding the centres. In terms of results gradient decent out performs the other 2 methods but in terms of time and computational effort it is more expensive. Other emerging methods like cooperative competitive genetic algorithms [32] for finding the centres have also been tried but needs further study.

Bibliography

[1] R. A. Brooks, "A robust layered control system for mobile robots," *IEEE Transaction on Robotics and Automation*, pp. 14–23, 1986.

[2] I. Noda, M. Hitoshi, K. Hiraki, and I. Frank, "Soccer server a tool for research on multi agent systems," *Applied Artificial Intelligence*, vol. 12, no. 2-3, 1998.

[3] P. Stone and M. Veloso, "A layered approch to learning client behaviors in the robocup soccer server," *Applied Artificial Intelligence*, vol. 12, 1998. http://www.cs.cmu.edu:80/afs/cs/usr/pstone/public/papers/97aai.ps.gz.

[4] P. Stone and M. Veloso, "Beating a defender in robotic soccer: Memory-based learning of a continuous function tech report: CMU-CS-95-222," tech. rep., CMU, 95. http://www.cs.cmu.edu:80/afs/cs/user/pstone/public/papers/MemBased-distribution.ps.Z.

[5] P. Stone and M. Veloso, "Using decision tree confidence factors for multiagent control.," in *Second International Conference on Autonomous Agents*, 1998. http://www.cs.cmu.edu:80/afs/cs/usr/pstone/public/papers/97springer/dt-paper/dt-paper.ps.gz.

[6] P. Stone and M. Veloso., "Towards collaborative and adversarial learning: A case study in robotic soccer," *International Journal of Human-Computer Systems*, vol. 48, no. 1, 1998. http://www.cs.cmu.edu:80/afs/cs/user/pstone/public/papers/96ijhcs.ps.gz.

[7] Milind Tambe and J. Adibi and Y. Al-Oraizan etal, "Building agent teams using explicit team work model and learning," *Artificial intelligence*, vol. 110, pp. 215–239, 1999.

[8] R. Sutton and A. Barto, *Reinforcement Learning an Introduction.* MIT press, 1998.

[9] D. E. Goldberg, *Genetic Algorithms in Optimisation and machine learning.* Addison Wessley, 1989.

[10] P. R. Kumar and P. P. Varaia, *Stochastic Systems: Estimation, Identification and Adaptive Control.* Prentice Hall.

[11] R. Sutton, "Planning by incremental dynamic programming," in *8th International Workshop on Machine Learning*, pp. 353–357, Morgan Kauffman, 1991.

[12] A. Moore and C. G. Atkeson, "Prioritised sweeping:reinforcement learning with less data in less real time," *Machine Learning*, vol. 13, pp. 103–130, 1993. http://www.cs.cmu.edu/afs/cs.cmu.edu/project/reinforcement/ papers/moore.prisweep.ps.Z.

[13] A. G. Barto, S. J. Bradke, and S. P. Singh, "Learning to act using real time dynamic programming," *Artificial Intelligence*, vol. 72, no. 1, pp. 1–138, 1995.

[14] C. W. A. A. G. Barto, R. S. Sutton, "Neuron like adaptive elements that can solve difficult problems," *IEEE SMC*, vol. 13, no. 3, pp. 834–846, 1983.

[15] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[16] L. P. Kaelbling, M. L. Littman, and A. P. Moore, "Reinforcent learning: A survey," *Journal of Artificial Intelligence Research (JAIR)*, vol. 4, pp. 237–285, 1996.

[17] J. Boyan and A. Moore, "Generalisation in reinforcement learning: Safely approximating the value function," in *Advances in Neural Information Processing Systems 7* (T. K. L. G. Tesauro,

D. S. Touretzky, ed.), vol. 7, pp. 369–376, Morgan Kauffman, 1995. http://www.cs.cmu.edu/afs/cs.cmu.edu/project/reinforcement/papers/boyan.funapprox-dp.ps.Z.

[18] R. S. Sutton, "Generalisation in reinforcement learning: Succesful examples using sparse coding," *Advances in neural information processing sytems*, vol. 8, 1996.

[19] L. Baird and A. H. Klopf, "Reinforcement learning with high dimensional continuous action," tech. rep., Wright Laboratory, Wright Patteson Air Base, 1993.

[20] J. Park and I. W. Sandberg, "universal approximation using radial basis functions networks," *Neural Computation*, vol. 3, no. 2, pp. 246–257, 1991.

[21] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Baltimore: Johns Hopkins University Press, 2nd ed., 1989.

[22] C. C. Lee, "Fuzzy logic in control systems: Fuzzy logic controller part i/ii," *IEEE SMC*, vol. 20, no. 2, pp. 404–435, 1990.

[23] J. S. R. Jang, "Self learning fuzzy controller based on temporal backpropagation," *IEEE SMC*, vol. 3, no. 5, pp. 714–723, 1992.

[24] J. A. Dickerson and B. Kosko, "Fuzzy function approximation with ellipsoidal rules," *IEEE SMC*, vol. 26, no. 4, pp. 542–560, 1996.

[25] T. L. Seng, M. B. Khalid, and R. Yousof, "Tuning of a neuro-fuzzy controller using genetic algorithms," *IEEE SMC*, vol. 29b, no. 2, 1999.

[26] T. Tagaki and M. Sugeno, "Fuzzy identification of systems and its application to modelling and control," *IEEE SMC*, vol. 15, pp. 116–132, 1985.

[27] N. B. Karayianis, "Growing radial basis neural networks: Merging supervised and unsupervised learning with network growth techniques," *IEEE trans neural networks*, vol. 8, no. 5, pp. 1492–1507, 1997.

[28] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data," *Journal of Royal Statistical Soceity*, vol. 39, no. 1, pp. 1–38, 1977.

[29] K. Deng and A. Moore, "Multiresolution instance based learning," in *International joint Conference on Artificial Intelligence IJCAI-95*, pp. 1233–1239, Morgan Kaufmann, 1995. http://www.cs.cmu.edu/afs/cs.cmu.edu/project/reinforcement/papers/deng.multires.ps.

[30] K. Mehlhorn, *Data Structures and Algorithms 3 : Multidimensional Searching and Computational Geometry.* Springer Verlag, 1984.

[31] A. McGovern and R. S. Sutton, "Macro actions in reinforcement learning technical report number 98-70," tech. rep., University of Massachusetts, 1998.

[32] B. A. Whitehead and T. Choate, "Cooperative competeitive genetic evolution of radial basis function centres and widths for time sereis prediction," *IEEE trans neural networks*, pp. 869–879, 1996.